A deep learning approach for automatically generating descriptions of images containing people

Trabajo de Fin de Grado
Curso 2017–2018

Autor
Marta Aracil Muñoz

Directores
Gonzalo Méndez Pozo
Raquel Hervás Ballesteros

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

A deep learning approach for automatically generating descriptions of images containing people

Trabajo de Fin de Grado en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Marta Aracil Muñoz

Directores
Gonzalo Méndez Pozo
Raquel Hervás Ballesteros

Convocatoria: *Septiembre 2018*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Abstract

Generating image descriptions is a challenging Artificial Intelligence problem with many interesting applications such as robots' communication or helping visually impaired people. However, it is a complex task for computers: it requires Computer Vision algorithms, to understand what the image depicts, and Natural Language Processing algorithms, to generate a well-formed sentence. Nowadays, deep neural networks are the state-of-the-art in these two Artificial Intelligence fields.

Furthermore, we believe that images that contain people are described in a slightly different manner and that restricting an image description generator model to these images may produce better descriptions. Therefore, the main objective of this project is to develop a Deep Learning model that automatically produces descriptions of images containing people and to conclude if it is a good practice the restriction to this kind of images. For this purpose, we have reviewed and studied the literature in the field and we have built, trained and compared four different models using Deep Learning techniques and a GPU to speed-up the computation, as well as a big and complete dataset.

Keywords

Deep Learning, Computer Vision, Natural Language Processing, image description generation, Keras, GPU, dataset.

Resumen

Generar descripciones de imágenes es un problema de Inteligencia Artificial con muchas aplicaciones interesantes como la comunicación de robots o ayudar a personas con discapacidad visual. Sin embargo, es una tarea compleja para un ordenador: requiere algoritmos de visión por computador para entender lo que la imagen representa y algoritmos de procesamiento de lenguaje natural para generar una frase bien formada. Hoy en día, las redes neuronales profundas son el estado del arte en estos dos campos de la Inteligencia Artificial.

Por otra parte, creemos que las imágenes que contienen personas se describen de manera ligeramente diferente y que restringir un modelo de generación de descripciones de imágenes a imágenes de este tipo puede producir mejores descripciones. Por lo tanto, el principal objetivo de este proyecto es desarrollar un modelo de aprendizaje profundo que produce automáticamente descripciones de imágenes que contienen personas y concluir si es una buena práctica la restricción a esta clase de imágenes. Para ello, hemos revisado y estudiado la literatura y hemos construido, entrenado y comparado cuatro modelos diferentes usando técnicas de aprendizaje profundo y una GPU para acelerar los cálculos, así como un dataset grande y completo.

Palabras clave

Aprendizaje profundo, visión por computador, procesamiento de lenguaje natural, generación de descripciones de imágenes, Keras, GPU, dataset.

Contents

List of Figures

Chapter 1

Introduction and objectives

As humans, we are constantly and unconsciously making descriptions of the world around us in order to communicate with others. We describe what we see in many different situations, such as when we give directions to someone (e.g., *go straight until you see the red building then turn right to the street with big trees and a fountain*), when we narrate a story to a friend (e.g., *I was shopping when I found Sara holding hands with a tall old man*) or when we post a picture on the Internet (e.g., *Having fun at the beach with my new blue dress!*). While we produce these descriptions easily and naturally without having to think too much about it, it is a very complex and challenging task for computers to automatically generate descriptions about the elements depicted in an image.

Producing a good description of an image involves first analyzing and understanding what appears in it and then generating a natural language sentence that must explain those aspects, a sentence that must be well-formed and concise. Therefore, generating image descriptions is more difficult than regular, well-known computer vision tasks such as image classification or object detection, as it entails not only Computer Vision (CV) techniques, but also Natural Language Processing (NLP), two Artificial Intelligence fields that have usually followed separate paths and are not typically studied and applied in combination.

Image description generation can be seen as a particular case of machine translation where, instead of translating a given sentence into another language, we need to translate an input image into its description. The input image must be *encoded* into a vector representation and then this representation must be *decoded* into a natural language sentence. To achieve this task, we can combine state-of-the-art results of both fields: Computer Vision results to obtain the vector representation of the image and NLP results (in particular, Natural Language Generation (NLG)) to generate the description.

This challenging Artificial Intelligence problem has been in the spotlight of big companies such as Facebook or Google. Automatically describing images is not only appealing because of its academic and scientific interest, but because of its many applications. The main and most important application is generating descriptions for the visually impaired,

either to help them in their daily life (for example, to move around the city or to buy in the supermarket), to bring culture closer to them by describing paintings in a museum or, as we live in a world clearly dominated and guided by images, to make social media, and web pages in general, more accessible to them. But it has other applications as well, such as robots' communication (transforming what they *see* into sentences and understanding who are they *talking* to) or medical image understanding.

On the other hand, in the last few years we have witnessed an impressive growing interest in obtaining data insights, and we can say that we are nowadays in what they call the *data-driven* era. Everyday, millions of powerful data are generated, many enterprises have great amounts of collected data and want to take advantage of it, uncovering hidden patterns and insights to produce meaningful business decisions. Due to this potent data availability and to the growing computational capacity to perform complex calculations, Machine Learning and Deep Learning techniques have been developed and widely studied and have become the state-of-the-art in many fields, what has lead to being on everyone's lips.

Therefore, the aim of this project is to cover these two attractive and popular topics by using Deep Learning techniques to develop an image description generator.

1.1 Objectives

Regarding image description generation, *Bernardi et al.* [Bernardi et al., 2016] distinguishes two main approaches: models that generate novel descriptions for a given image and models that build the description filling templates or retrieving descriptions associated to similar images; we will focus our work on the former.

We will restrict the scope of this thesis to generating descriptions of images containing people, as it is an interesting topic and we can focus all our efforts on it. Related works in automatic image description generation are not often focused on a particular image topic, trying to cover all possible descriptions and images, and they have not given too much attention to images that include people. We believe that we do not describe the same way people than we do with other entities, and so that restricting ourselves to images containing people could lead to better descriptions and results.

With all this in mind, the main objectives of this project are:

- Review and analyze the bibliography and the state-of-the-art techniques of generating image descriptions.
- Study and understand Deep Learning models and their applications.
- Study different Deep Learning frameworks and tools.
- Develop different models to automatically generate descriptions of images containing people and try to reproduce state-of-the-art results.

- Evaluate and compare all the developed models and determine which one is the best and if it is successful, to focus our efforts on and restrict our models to people's characteristics.

1.2 Document structure

The structure of this document will go as follows: Chapter 3 gives a thorough explanation of the state-of-the-art in both NLG and Computer Vision fields, and will also illustrate some general-purpose image description generators. In Chapter 4 we introduce the framework where we will develop our project; in particular, we explain GPUs, the cloud computing environment, the dataset election and which specific libraries for Deep Learning and visualization we will use.

Chapter 5 is the main chapter of this project; it goes step-by-step through all the models' definition and training processes. We explain in it all the characteristics and components of the models and all the decisions taken while defining and training them. Chapter 6 is dedicated to show the results obtained, the testing process and the most widely used metrics. Finally, in Chapter 7 we present our conclusions about this project and we give some possible future work directions.

Chapter 2

Introducción y objetivos

En nuestra condición de humanos, constante e inconscientemente hacemos descripciones del mundo que nos rodea para comunicarnos con la gente. Describimos lo que vemos en muchas situaciones diferentes, como por ejemplo cuando damos indicaciones a alguien sobre cómo ir a algún sitio (*sigue recto hasta que veas el edificio rojo y después gira a la derecha a la calle que tiene los árboles grandes y una fuente*), cuando le contamos una historia a un amigo (*estaba de compras cuando me encontré a Sara de la mano de un señor mayor alto*) o cuando subimos una foto a Internet (*¡Pasándomelo genial en la playa con mi vestido azul nuevo!*). Mientras que estas descripciones las hacemos de manera sencilla y natural, sin tener que pensarlo demasiado, generar automáticamente descripciones de lo que hay en una imagen es una tarea muy compleja y exigente para un ordenador.

Generar una buena descripción de una imagen requiere primero analizar y entender lo que aparece en ella y después generar una frase en lenguaje natural que explique esos elementos, esté bien formada y sea concisa. Por lo tanto, generar descripciones de imágenes es más difícil que las típicas tareas del campo de visión por computador, como son la clasificación de imágenes o la detección de objetos, ya que conlleva no solo técnicas de Visión por Computador (CV), sino también Procesamiento del Lenguaje Natural (NLP); dos campos de la Inteligencia Artificial que por lo general han llevado caminos separados y no se suelen estudiar ni aplicar en conjunto.

La generación de descripciones de imágenes se puede considerar como un caso particular de la traducción automática donde, en lugar de traducir una frase dada a otro idioma, hay que traducir una imagen de entrada a su descripción. La imagen de entrada debe ser codificada a un vector de representación y, después, esta representación debe ser decodificada a una frase en lenguaje natural. Para lograr esto, podemos combinar resultados del estado del arte de ambos campos: los resultados de Visión por Computador para obtener el vector de representación y los de NLP (en particular, el estado del arte de la Generación de Lenguaje Natural (NLG)) para construir la descripción.

Este problema tan interesante de Inteligencia Artificial ha despertado el interés de grandes empresas como Facebook o Google. El problema de describir imágenes au-

tomáticamente no solo es atractivo por su interés académico y científico, sino también por sus muchas aplicaciones. La principal y más importante aplicación es generar descripciones para las personas con alguna discapacidad visual, bien sea para ayudarles en su día a día (por ejemplo, para moverse por la ciudad o comprar en un supermercado), para acercarles a la cultura generando descripciones de los cuadros de un museo o, como vivimos en un mundo claramente dominado por fotos, para hacer más accesibles las redes sociales, y las páginas de Internet en general. Pero también tiene otras aplicaciones como por ejemplo en robótica para la comunicación de los robots (transformando en frases lo que *ven* e identificando con quién están *hablando*) o en medicina para describir imágenes médicas.

Por otro lado, en los últimos años hemos podido presenciar un increíble aumento del interés en obtener información de los datos y estamos actualmente en lo que llaman la era *data-driven* (dirigida por los datos). Cada día, se crean millones de nuevos datos, muchas empresas tienen recogida una gran cantidad de datos y quieren sacarles provecho, descubriendo patrones ocultos y perspectivas para obtener buenas decisiones de negocio. Debido a esta gran cantidad de datos disponibles y al crecimiento de la capacidad computacional para realizar cálculos complejos, las técnicas de aprendizaje automático y aprendizaje profundo se han podido desarrollar y estudiar convirtiéndose en el estado del arte en muchos campos, lo que ha hecho que estén en boca de todos.

Por lo tanto, este trabajo busca aunar estos dos conceptos tan populares y atractivos, desarrollando un generador de descripciones de imágenes utilizando técnicas de aprendizaje profundo.

2.1 Objetivos

Respecto a las descripciones de imágenes, *Bernardi et al.* [Bernardi et al., 2016] distinguen dos enfoques principales para su generación: modelos que generan descripciones nuevas para una imagen de entrada y modelos que construyen la descripción rellenando plantillas o usando descripciones asociadas a imágenes similares; nosotros centraremos nuestro trabajo en los primeros.

Vamos a limitar el alcance de este trabajo a generar descripciones de imágenes de personas, ya que consideramos que es un enfoque interesante y podremos centrar todos los esfuerzos en ello. El trabajo relacionado en el campo de la generación automática de descripciones de imágenes no se suele centrar en un tema en concreto, tratando de abarcar todas las posibles descripciones e imágenes y no le han dado demasiada atención a las imágenes de personas. Consideramos que no describimos de la misma manera a una persona que a otros elementos de una imagen, y por tanto, al restringirnos a imágenes de personas podemos obtener mejores descripciones y resultados.

Con todo esto, los objetivos principales de este trabajo son:

- Revisar y analizar la bibliografía y el estado del arte de las técnicas de generación

de descripciones de imágenes.

- Estudiar y entender los modelos de aprendizaje profundo y sus aplicaciones.
- Estudiar los diferentes entornos de trabajo y herramientas enfocadas al aprendizaje profundo.
- Desarrollar diferentes modelos que generen automáticamente descripciones de imágenes de personas y tratar de reproducir los resultados del estado del arte.
- Evaluar y comparar los distintos modelos creados y determinar cuál es el mejor y si es satisfactorio centrar los esfuerzos y limitar nuestros modelos a características de personas.

2.2 Estructura del documento

La estructura de este documento será la siguiente: el Capítulo 3 da una explicación muy detallada del estado del arte tanto en el campo de la Generación de Lenguaje Natural como en el de Visión por Computador, en este capítulo también se detallan algunos generadores de descripciones de imágenes. En el Capítulo 4 introducimos el entorno de trabajo en el que desarrollaremos este trabajo; en particular, explicamos las GPUs, el entorno *cloud computing*, qué librerías para aprendizaje profundo y visualización usaremos y la elección del dataset.

El Capítulo 5 es el principal de este trabajo; va paso a paso por el proceso de definición y entrenamiento de los modelos. Explicamos en este capítulo todas las características y componentes de los modelos y todas las decisiones tomadas al definirlos y entrenarlos. El Capítulo 6 está dedicado a mostrar los resultados obtenidos, el proceso de testing y las métricas más conocidas. Por último, en el Capítulo 8 presentamos las conclusiones del trabajo y damos posibles líneas de trabajo futuro.

Chapter 3

State of the art

Image analysis has been a challenging problem in researchers' minds for a long time now. At first, this problem was tackled by a traditional computer vision approach, with explicit processing algorithms. Then, Machine Learning models appeared and they were the state-of-the-art in this field until 2010, when training complex deep neural networks started to become a possibility thanks to the use of GPUs computational power and the availability of more data. As of today, these neural networks, in particular Convolutional Neural Networks, are the state-of-the-art in all image analysis challenges.

Similarly, the task of Natural Language Generation (NLG), has been faced by classical step-by-step algorithms, as well as Machine Learning and Deep Learning models. In particular, Recurrent Neural Networks are, as of today, the state-of-the-art in generating natural language sentences and texts.

Image description combines both computer vision and natural language processing fields, and, nowadays, the state-of-the-art results are a combination of Convolutional Neural Networks, that focus on extracting image features, and Recurrent Neural Networks, to generate the sentence.

In this chapter we will present previous work in the field of image analysis and Natural Language Generation, and how these fields have been addressed across time, from the point of view of traditional and explicit algorithms to the one of Deep Learning models. We start with a brief introduction to traditional algorithms for computer vision and language generation, we then explain the main characteristics of Machine Learning and we continue with Deep Learning. In the last section of this chapter, we discuss some well-known image caption generators.

3.1 Traditional algorithms for image analysis

Image analysis consists of the extraction of important characteristics and features of images by digital image processing techniques, and it has many different fields of application, such as robotics, security, medicine or biology.

Traditional computer vision algorithms, in general, work by extracting feature vectors from images and using them to classify images. There are algorithms responsible for a particular task (noise reduction, image segmentation, corner detection, edge detection...), that then work together with other specific algorithms to carry out the whole image processing task. The use of these techniques to extract useful structural information from images, such as edges, corners or colors, reduces significantly the amount of data to be processed, as it filters out non-relevant data to focus only in the useful information extracted.

The main feature detection algorithms in computer vision are:

- Canny edge detector [Canny, 1986]. An edge is a sudden change in image brightness, that is, a point where the image brightness has discontinuities. These discontinuities usually correspond to changes in surface orientation, depth, material properties or scene illumination. Therefore, an edge detector may help to identify the boundaries of objects and surface markings, as well as variations in surface orientation. The Canny edge detector is one of the best edge detector algorithms, and can detect a wide variety of edges in an image. It first smooths the image by applying a Gaussian filter, then finds intensity gradients and selects the potential edges.

- Harris corner detector [Harris and Stephens, 1988]. A corner is interpreted as the intersection of two edges. It is a very important image feature as it is a point invariant to translation, rotation and illumination. The Harris corner detector algorithm focuses on the detection of corners in an image, reducing the dimensionality of data to be processed and it is used in many computer vision applications such as motion tracking or stereo vision (extracting 3D characteristics from images). It has been proved to be one of the most accurate algorithms in distinguishing between edges and corners.

- SIFT (Scale-Invariant Feature Transform) [Lowe, 1999]. It is a feature description algorithm used to detect and describe features in images. SIFT features are invariant to image location, scale and rotation. A descriptor vector is computed for each one of the points of interest in the image, so that they are also robust to changes in noise and illumination. This algorithm is useful for many computer vision applications, such as object recognition, motion tracking, navigation or 3D modeling.

- SURF (Speeded-Up Robust Features) [Bay et al., 2006]. It is also a feature extractor and descriptor. This algorithm is a speeded-up version of SIFT.

These object detection algorithms usually come after a process of image segmentation,

that is, dividing the image into its constituents parts, and are then combined with traditional Machine Learning algorithms (SVM or K-Nearest neighbor) for image classification.

3.2 Traditional approach to NLG

Natural Language Generation (NLG), involves producing understandable natural language texts from some input data. It has many different interesting applications, besides generating image descriptions, such as machine translation, robotics or textual summaries of advanced databases (e.g., financial or weather forecasts datasets) [Goldberg et al., 1994, Iordanskaja et al., 1992, Wu et al., 2016].

Traditional NLG algorithms are based on the field of formal language theory and logic rules on which we can build the language. As Reiter and Dale [Reiter and Dale, 1997] state, traditional algorithms in NLG usually follow these steps:

1. Content determination: deciding which information should be included in the output. This process creates a set of *messages* from the input data, and it is typically done by filtering and summarizing the data and defining the *messages* in some formal language, usually application-dependent.

2. Document structuring: organizing and structuring the set of *messages* so that the final text makes sense. The output of this step is usually represented as a tree structure.

3. Aggregation: merging related *messages* into sentences to compact the information. This step is not mandatory but enables fluency and readability.

4. Referring Expressing Generation: producing expressions identifying objects that the text refers to, as well as deciding about pronouns. It is related to lexicalization.

5. Lexicalization: selecting the specific words and phrases to be used to represent concepts and relations that appear in the *messages*.

6. Realization: applying rules of syntax, morphology and spelling to produce the final output, such as adding prepositions, making plurals or adding punctuation marks.

3.3 Machine Learning

Machine Learning is the sub-field of Artificial Intelligence that studies computer algorithms that improve automatically through experience. In contrast to traditional algorithms, that have specific rules for doing the tasks they are required for, Machine Learning algorithms *learn* concepts without being explicitly programmed for doing so, identifying patterns from given examples to perform accurately on new, unseen data by inferring these uncovered

patterns. These algorithms can learn from and make predictions on large volumes of data, not exclusively images.

The field started to be studied in the mid 1980s and early 1990s, and became more popular around 2010 as more and more data were available for learning and training the Machine Learning models.

This discipline has many applications in a wide variety of fields where designing and programming explicit algorithms with good performance is difficult or infeasible. These applications include voice recognition, natural language processing, translation, search engines, computer vision, robotics, medical diagnosis, financial market analysis, advertising or recommender systems [Pang et al., 2002, Wernick et al., 2010, Bridge et al., 2014, Sarikaya et al., 2014, Baik and Bala, 2004].

We can classify Machine Learning techniques into two categories: supervised learning and unsupervised learning. In supervised learning, the Machine Learning algorithm is given a set of sample inputs together with their desired outputs, so that it can train with those examples to infer a generalized function that maps inputs into outputs and predicts well when given a new example. Therefore, already labeled-data are required in this approach.

There are also some special cases of supervised learning such as semi-supervised learning, that consist of a set of examples where not all of them go with their corresponding desired output, that is, there are labeled and unlabeled data on the set; active learning, where the algorithm has only a small initial set of labeled data and it is able to interactively make queries to the user to find the desired outputs for some unlabeled data; and reinforcement learning, where there is some sort of feedback obtained from the outside in a dynamic environment as an answer for its actions.

Some of the main supervised learning algorithms and approaches are the following:

- Decision trees and random forests: they are used as a predictive model. In a decision tree, the branches represent the observations about an item, its features and the leaves represent class labels. Random forests [Ho, 1995] construct many decision trees to make more accurate predictions by outputting a combination of the outputs of all those decision trees, typically the mode or mean. Random forests are very popular as it is easy and clear to see and understand the decisions taken by them, while other Machine Learning algorithms are more obscure.

- Logistic regression: it is a statistical method to model a binary dependent variable (classes 0 and 1) in terms of one or more explanatory variables, using the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$. This function returns the probability of belonging to class 1. The algorithm's output is then selected by comparing the probability to a given threshold (usually 0.5).

- Support Vector Machines (SVM) [Cortes and Vapnik, 1995]: it is a binary classification algorithm that given a set of points of two different classes in a n-dimensional

space, separates them with the hyperplane that is situated the furthest from all the points. This algorithm is only useful if the points are linearly separable, if not, the algorithm was improved by including kernels, that project the set of points in a bigger dimension space, to obtain a hypersurface that separates the points in the initial space.

- Neural networks: it is a learning algorithm where computations are structured in connected neurons that transmit signals from one to another. Therefore, each neuron's input is the output of another neuron (or the input of the algorithm for the first layer of neurons), and each neuron's output is computed as a non-linear function of the input.

- Nearest neighbor algorithm [Altman, 1992]: it is used both for classification (there are a discrete number of classes) and regression (the target variable is continuous), and it is based on the information of the k closest points. In classification, the output is the most common class among its k neighbors, and in regression the output is the mean of the values of these neighbors.

Unsupervised learning algorithms learn from a set of unlabeled data, input examples without their desired output, inferring a function that can describe the structure in the data. Therefore, the system must be capable of finding patterns in the data in order to be able to label the new inputs; but, as the data used to train the algorithm is unlabeled, there is no way of evaluating the accuracy of the patterns found in it.

The following are the most common unsupervised learning algorithms:

- Clustering: these methods divide the data set into different subsets, or clusters, so that the points in the same cluster are more similar between them than between those of other groups. The goal is to maximize the similarity between points in the same subset and to maximize the difference between points of different subsets.

- Principal Component Analysis (PCA) [Hotelling, 1933]: this algorithm is used to reduce the dimensionality of the data set, without losing any important information and filtering redundancy, by making linear combinations of the original data variables and selecting the ones that have the largest possible variance and are uncorrelated.

3.4 Deep Learning

Deep learning is a Machine Learning technique, based on neural networks with many layers between the input and the output ones [LeCun et al., 2015].

Although the theory behind Deep Learning has been developed for many years now, deep neural networks need much computer capacity and labeled data to perform rapidly and correctly the complex tasks it is useful for. Therefore, Deep Learning has only recently

become more popular because of the development of GPUs for general purposes and the increase in the size of datasets.

While the performance of Machine Learning algorithms usually converge at some point, not increasing if more training data is supplied, Deep Learning networks are scalable with data. This allows Deep Learning to formulate more complex models than Machine Learning. These models can be used in many complex applications of image recognition and natural language processing, where they usually are the state-of-the-art, even with better performance than humans [Krizhevsky et al., 2012, Graves et al., 2013, Wu et al., 2016].

Artificial neural networks, motivated by biological neural networks, constitute a collection of connected neurons, structured in layers, where the information travels from the input layer to the output layer, going through the hidden layers. Each neuron computes a non-linear function of its inputs and transmits the output to the neurons it is connected to, where the received information is the input of that neuron. Deep neural networks are artificial neural networks with a great number of hidden layers.

In particular, the simplest networks are feedforward neural networks, explained in Section 3.4.1. Moreover, within all neural networks models that have been studied during these years, there are two particular networks, Convolutional Neural Networks and Recurrent Neural Networks, that are the state-of-the-art for computer vision and for natural language processing, respectively.

However, Deep Learning networks are usually slow to train, especially with image datasets. There are two well-known techniques frequently used to speed-up the training process: using already pre-trained Deep Learning networks and fine-tuning. Pre-training takes advantage of already computed neural network weights to use them as part of a bigger neural network for a similar problem; fine-tuning re-trains, starting with its pre-computed weights, an already trained neural network with a more specific dataset of similar characteristics.

3.4.1 Feedforward Neural Networks

The simplest neural network is the feedforward neural network, where the connections between neurons do not form any cycle, and so the information goes forward from the input layer to the output layer. Every neuron is connected to all of the neurons in the next layer and every connection is labeled with a certain weight ω. Figure 3.1 shows this neural network general representation.

Each neuron computes its output o as a non-linear function f, called activation function, of the sum of the weighted inputs and a bias b:

$$o = f(\omega^T x + b)$$

The most commonly used activation functions are the sigmoid function, the hyperbolic tangent and the ReLU (Rectified Linear Unit) function. These functions are shown in Figure 3.2.

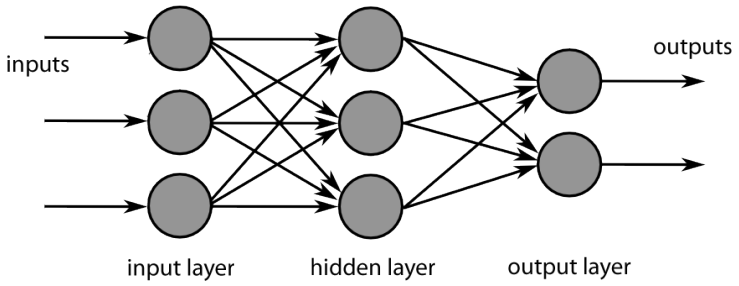


Figure 3.1: Feedforward Neural Network

However, these weights and biases are the model parameters; thus, the model must be trained in order to compute the weights and biases values that make the neural network more accurate when given new data. The optimal values for these parameters are the ones that minimize the cost function that compares the output of the model to the expected output.

The gradient descent method is an optimization algorithm for approximating a local minimum of a differentiable function f (the global minimum if it is a convex function). It is used in Deep Learning while training, in order to obtain a good estimation of the model parameters x. As the goal is to minimize the function, this method moves iteratively across the parameters' space in the direction that decreases the function the most; this direction happens to be the negative of the gradient of the function. The parameters are then updated in each iteration by a factor α, called learning rate, of this direction until it converges to a local minimum:

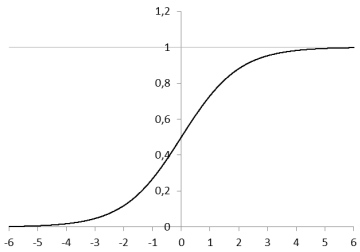
$$x = x - \alpha \nabla f(x)$$

The backpropagation algorithm is an efficient way of computing these gradients in neural networks, as they can be represented by a computation graph as we have shown in Figure 3.1. This algorithm starts in the output layer and goes backwards in the graph using the chain rule to compute the gradients of each layer's parameters.

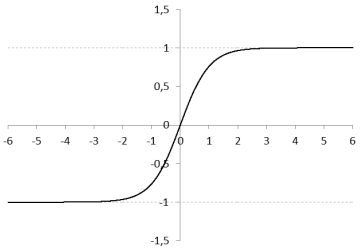
3.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are one kind of artificial neural networks where neurons of each layer do not necessarily have to be connected to every neuron of the next layer and weights are shared across the layer. This architecture provides local processing, learning local features of the input, and it is space-invariant, recognizing features no matter where they are located in the input. Thus, these neural networks are basically and efficiently used for image processing [Donahue et al., 2015, Karpathy and Li, 2015, Krizhevsky et al., 2012, Tran et al., 2016, Vinyals et al., 2015].

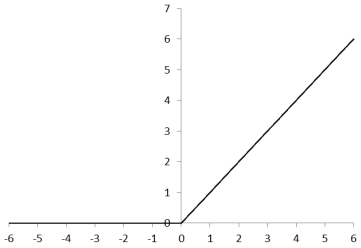
Moreover, they reduce considerably the computational cost, since they need to learn less number of parameters, making convolutional neural networks scale to high resolution



(a) Sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$.



(b) Hyperbolic tangent $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.



(c) ReLU function $f(z) = max(0, z)$.

Figure 3.2: Regular activation functions

images. They also increase their performance, as they consider the spatial structure of the input in contrast to feedforward neural networks. CNNs are biologically inspired by the visual cortex, where visual neurons are structured such that each of them only focuses on a particular region of the visual field, called the receptive field, and the union of all receptive fields covers the whole visual field.

There are three different types of layers used in convolutional neural networks: convolutional layers, pooling layers and fully-connected layers. A typical CNN architecture is shown in Figure 3.3. Convolutional layers are the core of these networks, neurons inside these layers are connected only to some neurons of the previous layer. The output in these layers, called feature map, is computed by applying a non-linear function (typically ReLU function) to the convolution of the input and the shared weights, that is, it slides a local filter across the whole input. Filters are local in width and height but cross all the color channels of the image (three channels if it is a RGB image or only one if it is a greyscale image). Each convolutional layer usually applies more than one filter, obtaining more than one feature map, allowing more modeling capacity.

Pooling layers apply a non-linear function across their input, summarizing the information to reduce its dimensionality and resolution, by combining a region of pixels in an image (or a spatially local information in another type of input data) into a single one. The typical pooling layer is called max-pooling, which takes the maximum of all values in the region. This is done by sliding a small filter across the input, with overlapping or non-overlapping stride, but only for a single color channel of the image. A pooling layer comes usually after a few convolutional layers.

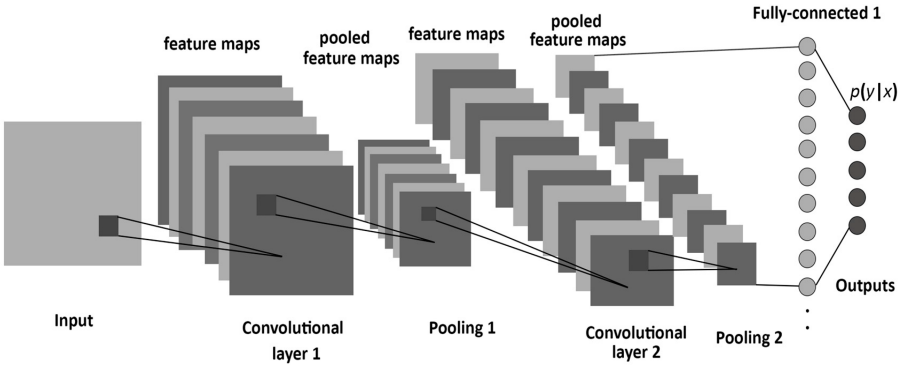


Figure 3.3: Convolutional Neural Network

Finally, after a collection of convolutional and pooling layers, there are some fully-connected layers at the end of the network. In these layers, neurons are connected to all neurons in the next layer, like in feedforward neural networks, and classify the input based on the features extracted in the previous layers.

3.4.3 Recurrent Neural Networks

Neural networks where connections between neurons can form cycles are called Recurrent Neural Networks (RNN). Because of these cycles, the output's computation can rely on previous computations: while for Feedforward Neural Networks every input and output are independent, not being able to distinguish relationships between them, cycles in Recurrent Neural Networks allow them to have *memory* of what they have previously *seen* or *said*. Therefore, they are essentially useful for models where the input or output has some sort of temporal dependencies, that is sequential information, such as speech recognition or natural language processing. RNNs can be used to model sequential input into sequential output, sequential input into temporal-independent output or temporal-independent input into sequential output, the latter is the case of image description generation.

By unfolding or unrolling the loop in the Recurrent Neural Network, it is easier to observe the sequential capability it has. Figure 3.4 shows RNNs' architecture and its unfolded version. The Recurrent Neural Network can be seen when it is unfolded, as a deeper traditional feedforward network with same weights for each layer, that is, weights are shared across time steps, reproducing the same calculations for all elements in the sequence. In Figure 3.4 U, V and W are the weights that are shared between steps; x_t and o_t are the input and output, respectively, in time step t and h_t, called *hidden state*, is the *memory* of the network, computed as a non-linear activation function f of the previous hidden state and the current input. The hidden state and the output are calculated as follows:

$$h_t = f(Ux_t + Vh_{t-1})$$

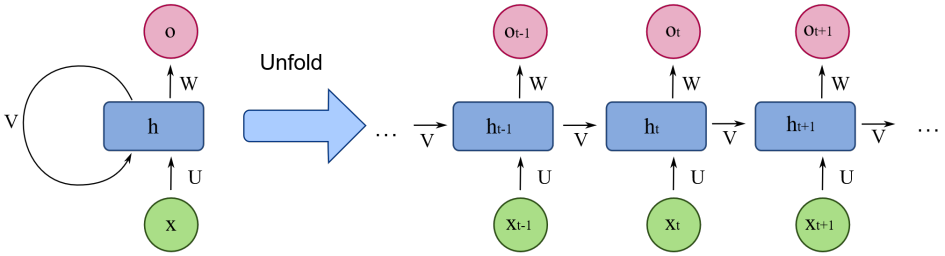


Figure 3.4: Recurrent Neural Network

$$o_t = g(W h_t)$$

The unrolled network in Figure 3.4 is the general sequence-to-sequence approach. If the input is a sequence but the ouput is *static* instead, we may only need the final output, the output of the last *layer* in the unfolded version. Similarly, if the output is a sequence but the input is *static* (as it is our case generating image descriptions), we may only need to pass the input to the network once, at the beginning of the recursion.

Recurrent Neural Networks are also trained using backpropagation, but a generalized version of it, called Backpropagation Through Time (BPTT) [Mozer, 1989]. BPTT algorithm calculates the gradients using the unfolded version of the network, as it also depends on the calculations of previous time steps. However, training with BPTT has some problems as, at each time step, gradients tend to grow or decrease which derives in vanishing or exploding gradients after some time steps, making it impossible to learn long-term dependencies [Bengio et al., 1994].

More complex variants of Recurrent Neural Networks architectures have been developed to try to solve these and other problems. As of today, there are many different RNNs options: *vainilla* RNNs (traditional RNNs), Hopfield networks, Long Short-Term Memories (LSTM), Gated Recurrent Units (GRU), bidirectional RNNs, echo state networks, Neural Turing Machines... The most popular Recurrent Neural Networks for NLP tasks are GRU [Cho et al., 2014] and LSTM [Hochreiter and Schmidhuber, 1997] which are equivalent ([Chung et al., 2014]) and introduce a memory unit to learn long-term dependencies.

3.5 Image description generators

In the last years, image description generators and, in general, Deep Learning applied to classify and describe images, have arisen. In this section we describe some of them.

As we have introduced in Chapter 1, the first approach in generating image de-

scriptions did not try to achieve novel descriptions, instead it was done with traditional algorithms by filling natural language templates based on what was observed in the image [Li et al., 2011, Yang et al., 2011, Mitchell et al., 2012, Kulkarni et al., 2013] or by retrieving and combining descriptions of similar images [Kuznetsova et al., 2012, Ordonez et al., 2011, Mason and Charniak, 2014].

Kiros et al. [Kiros et al., 2014a, Kiros et al., 2014b] were the first ones to tackle this problem using deep neural networks; their solution uses a feedforward neural network for generating novel descriptions. Another image description generator, that also uses deep neural networks, is Show and Tell [Vinyals et al., 2015], developed by Google Brain team in 2015. Show and Tell generates image descriptions by first extracting features with a Convolutional Neural Network and then generating understandable sentences with a LSTM Recurrent Neural Network. As *Vinyals et al.*, many others have also developed their models combining CNN for image understanding and LSTM for natural language generation [Mao et al., 2014, Donahue et al., 2015, Karpathy and Li, 2015, Chen and Zitnick, 2015, Hendricks et al., 2016].

CaptionBot[1] is an image caption generator developed by Microsoft Cognitive Services in 2016 that uses Deep Learning techniques for computer vision and natural language generation to describe image content automatically. This caption generator uses Convolutional Neural Networks for extracting the image features and recurrent neural networks for generating the description [Tran et al., 2016]. CaptionBot is available online, for everyone to upload any image and it returns a description of what the algorithm *sees* in it. It can also detect emotions if there is a person in the image. This description can be then rated so that the algorithm can improve its learning. Figure 3.5 shows an usage example of CaptionBot with a picture of a kid extracted from Google Images.

Google has also developed Google AIY Vision Kit[2], a do-it-yourself hardware kit to build an image recognition system that can identify objects. A Raspberry Pi and a Raspberry Pi Camera are needed to build this vision system.

Automatic Alternative Text [Wu et al., 2017b] is a new Facebook tool developed in 2016 for making images more accessible to people with visual deficiencies. It generates an automatic description of a Facebook photograph. This tool is based on Deep Learning Convolutional Neural Networks to extract photograph features. To generate the text description the model just lists the people, objects and scenes detected by the Convolutional Neural Network in that order.

Finally, a sophisticated wearable device for those blind or visually impaired, called Horus[3], will be released in the near future. Horus is equipped with a camera and a processor, that is able to read text, recognize objects and identify faces, and describe them to the user, by communicating him what or who it sees, using bone conduction[4].

[1] https://www.captionbot.ai/
[2] https://aiyprojects.withgoogle.com/vision/
[3] http://horus.tech
[4] Sound sent directly to the inner ear through the bones.

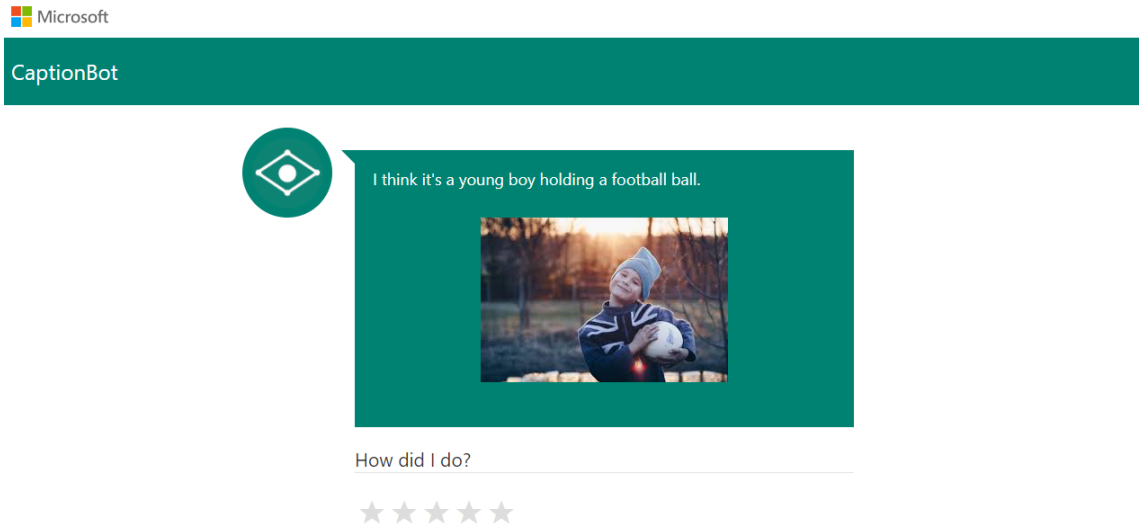


Figure 3.5: Captionbot

This device is even able to learn new faces and objects if the user shows them to Horus in different angles and speaking out loud the name of the object or the person.

As we have seen, more and more sophisticated image caption generators, for different purposes and contexts, are being developed, showing the interest and impact of this brand-new field and resulting in a rich literature.

Chapter 4

Framework

Training and developing Deep Learning models requires great computational capacity and a good election of the dataset. In this chapter, we explain what are GPUs and why are they useful for developing Deep Learning projects, which Deep Learning framework and visualization tools we have selected and which dataset we use. We developed the project using Python as programming language.

4.1 GPU

GPUs (*Graphics Processing Unit*) were traditionally created for computer graphics and image processing to accelerate the complex calculations of real-time 2D and 3D graphics that resulted in long processing time in CPUs.

GPUs are build to optimized tasks based on the SIMD parallel concept (*Single Instruction Multiple Data*), that is, the same instruction applied repeatedly over a bunch of data. Those repeated applications of the same instruction are all independent and can, therefore, be run simultaneously. Linear algebra operations are inherently parallel and are the basis of polygon transformations for scene rendering in computer graphics.

There are other power and time consuming applications of parallel nature that could be optimized by using a GPU, but they involve reformulating the problem into graphics primitives to be able to use them, which is a very complex task. For that reason, NVIDIA developed CUDA (*Compute Unified Device Architecture*), a parallel computing toolkit similar to C programming language, to exploit GPUs capabilities for general purposes, which is commonly called GPGPU (*General Purpose Graphics Processing Unit*).

Linear algebra is the basis of Deep Learning algorithms, and also many Machine Learning algorithms. Neurons within a layer apply all the same function to different input data, so they could be also parallelized and hence GPUs are always used for training these models.

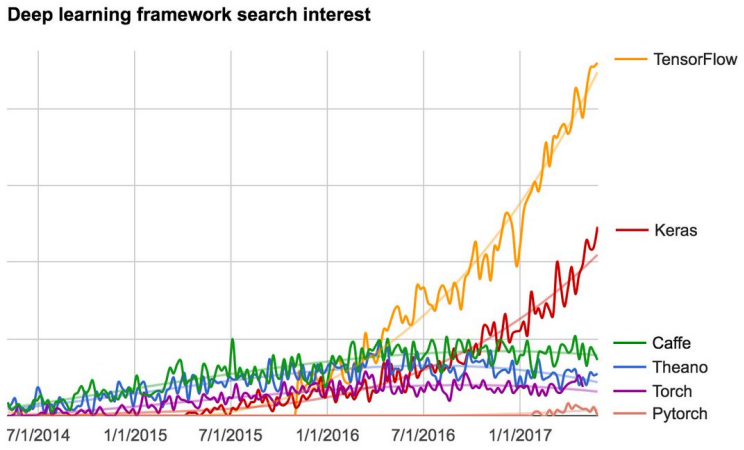


Figure 4.1: Deep Learning frameworks popularity

Acquiring a GPU is not cheap, there exists cloud computing platforms, such as Amazon Web Services[1], Google Cloud Platform[2] or Microsoft Azure[3] that contain different servers with several GPUs to allow users to run their models more economically. GAIA research group has built a server with an NVIDIA Titan X GPU in order to maintain Deep Learning projects for students and professors and gave us access to it, so we will use it for training this project.

4.2 Deep Learning framework

CUDA has a very low level API and makes implementing Deep Learning computational graphs considerably tedious. For this reason, many higher level frameworks were created for easily describing and developing the complex tasks that Deep Learning involves.

There is a wide number of frameworks available for Deep Learning training, the majority of them open-source and developed by big companies or university teams such as Microsoft, Google, Intel or Berkeley University. The most well-known and frequently used frameworks are Torch [Collobert et al., 2011], Tensorflow [Abadi et al., 2016], Caffe [Jia et al., 2014] and Theano [Theano Development Team, 2016].

Among all these frameworks, Tensorflow is probably the most popular one (as Figure 4.1, extracted from Google images, shows), used in distinguished companies such as Airbnb, Uber, Dropbox or Twitter. It is an open-source framework for advanced numerical computation, developed by the Google Brain team in 2015, replacing their propietary

[1] https://aws.amazon.com
[2] https://cloud.google.com/
[3] https://azure.microsoft.com

```
from keras.models import Sequential
from keras.layers import Dense

# Building the model and learning configuration
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['accuracy'])

# Training, evaluation and prediction
# x_train, y_train, x_test, y_test and new_data are numpy arrays
model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
prediction = model.predict(new_data, batch_size=128)
```

Figure 4.2: A simple Keras example code

Machine Learning system DistBelief [Dean et al., 2012], created in 2011. Tensorflow is widely used for Deep Learning and Machine Learning and it is optimize to run models both on CPUs or GPUs. It provides an API for Python as well as for other programming languages (C++, Haskell, Java...), but the most developed one is the Python library.

Tensorflow works well for modeling Convolutional Neural Networks and Recurrent Neural Networks. The basic object in Tensorflow library is the *tensor*, an abstraction of multidimensional arrays, being a 1-dimensional tensor a vector and a 2-dimensional tensor a matrix. Deep Learning computational graphs are build by describing tensors' operations and gradients are calculated automatically using a specific function.

However, programming in Tensorflow can be difficult and there exists a higher level API that allows flexible Tensorflow implementations in a more user-friendly manner. This API is Keras[4], a high-level open-source Python library for developing Deep Learning projects more easily and readable, created in 2015 by a Google engineer and released under the MIT license. Keras works with Tensorflow, as well as Theano or CNTK [Seide and Agarwal, 2016], as backend. In 2017 Tensorflow included it as part of its library and selected it as the preferred high-level API.

The basic structure in Keras is the *model*, a set of connected layers. Computational graphs are built by stacking layers with the `add()` function, where the user only needs to specify the layers and the inputs' dimensions. Keras library provides the well-known layers but allows users to also create new ones. Configuring the training process is as simple as applying to the model the `compile()` function with the specific conditions (loss function, optimizer, metrics...). Training, evaluating the model and making predictions on unseen examples is done with just other high level functions: `fit()`, `evaluate()` and `predict()`

[4]https://keras.io/

respectively. Figure 4.2 shows a simple example, extracted from Keras documentation, of how to use these functions to build a neural network with two fully-connected layers.

Therefore, as our needs are covered in Keras' library, it is widely recommended for Deep Learning starters and it is more user-friendly, we decided to develop our model in Keras on top of Tensorflow. Keras and Tensorflow installation is done easily in Linux via the `pip install` command, creating previously a Python virtual environment.

4.3 Visualization: TensorBoard

Deep Learning models are composed of millions of complex calculations that often complicate, or even make it impossible, to debug and understand the parameters and metrics, what happens during training and what is the neural network learning. For this purpose, Tensorflow developed Tensorboard[5], a suite of visualization tools inside Tensorflow library that allows users to visualize computation graphs, metrics results and other useful charts.

TensorBoard works by saving log files, containing execution information about the training process and the computation graph, into a specific log directory, and opening TensorBoard web server to access the graphic interface.

The desired data is obtained with *summary operations*, tensor's operations that produce serialized information about a model that needs to be read with TensorBoard. These summary operations receive the tensor we want to visualize and a meaningful name to distinguish all of them. Then, all the summaries created are combined using `tf.summary.merge_all` into a single operation that generates all of them. Finally, the serialized summary is written to the specified directory in disk with `tf.summary.FileWriter`.

To access the graphic interface of TensorBoard to visualize the summaries, we just need to go to `http://localhost:6006` after typing the following command in the command line:

```
$ tensorboard --logdir path/to/log-directory
```

It is also possible to compare visualizations of different executions by specifying the paths to the log directories in the previous command. The appearance of TensorBoard graphic interface is shown in Figure 4.3

TensorBoard contains a wide variety of helpful visualizations: `tf.summary.scalar`, `tf.summary.histogram`, `tf.summary.audio` or `tf.summary.image`, among others. The first one is the most used, it displays the variation of a scalar metric over time, usually the loss or the learning rate. All these visualizations admit some dynamic interactions within the TensorBoard interface.

Using TensorBoard within Keras high-level API it is even easier. It is just necessary to include the `callback_tensorboard` function while training, specifying the log directory.

[5] https://github.com/tensorflow/tensorboard

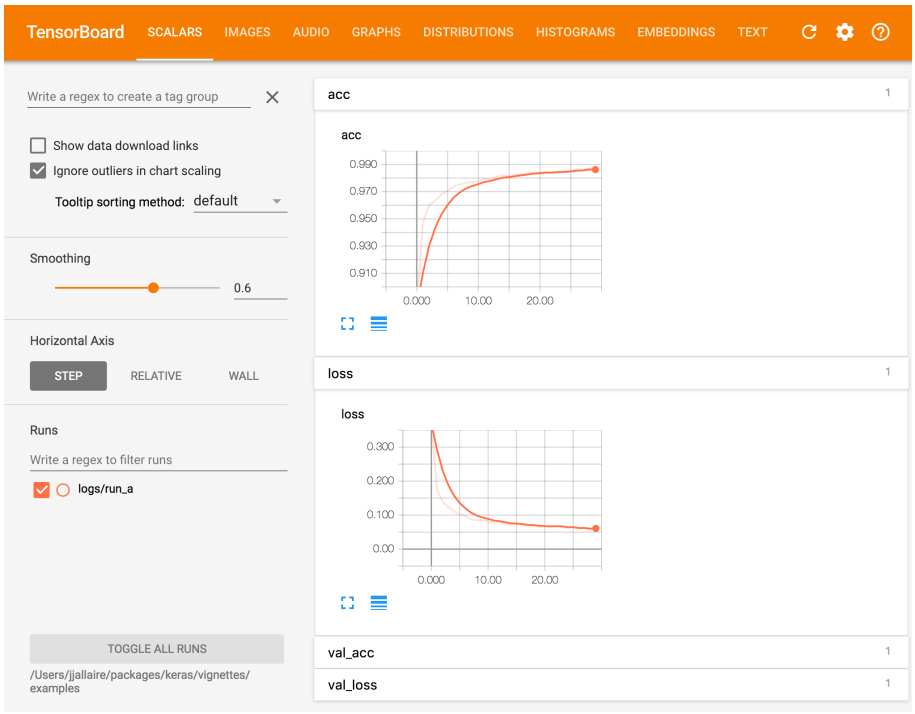


Figure 4.3: TensorBoard web server

This function will record data for loss and accuracy metrics, as well as other metrics specified in the `compile` function.

4.4 Image datasets

For correctly training and accurately evaluating a good Machine Learning model, a suitable selection or elaboration of the dataset is crucial. Once we have the dataset, it is divided into three disjoint subsets: training, test and validation sets. Training set is used for training the model, fitting the model's parameters; test set is used once the model is trained, to provide an unbiased evaluation of this final model; validation set, also called development set or *dev* set, is used while training the model, to tune the hyperparameters and to prevent overfitting.

As Deep Learning researchers have a strong interest in the image analysis field, during these years a lot of large and medium scale datasets have been created for training these models and we can find many of them available online. The most popular ones are explained below.

ImageNet

ImageNet[6] is probably the most well-known image dataset in the Deep Learning and machine learning industry. This image dataset is used mainly for classification purposes and the classes are organized based on the nouns in the WordNet [Miller, 1995] hierarchy.

This image database was created in 2009 [Deng et al., 2009] by querying image search engines with WordNet's synsets. They extracted a great number of candidate images for each synset and then filtered them by crowd-sourcing in Amazon Mechanical Turk (AMT)[7]. Since 2010, ImageNet also runs the ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) annual visual challenge [Russakovsky et al., 2015]. Nowadays, this challenge is hosted in Kaggle[8] and it has three categories: image classification for 1000 classes, object detection (detect the bounding box where the object lays) for 200 classes and object detection from video for 30 classes.

As of today, ImageNet is one of the largest image database available. It has a total of 14,197,122 annotated images of a wide variety of categories, with an average of over five hundred images per synset. For the person's high-level category (the one we have interest in), it holds around 952,000 images and it is divided in 2,035 synsets with an average of 468 images per synset. Images are annotated with their classification and 1,034,908 of them also with their bounding boxes.

MS COCO

MS COCO[9] (*MicroSoft Common Objects in COntext*) dataset [Lin et al., 2014] is one of the most widely used large-scale datasets for image description generation and also for object detection, object segmentation and person keypoints detection.

Currently, the MS COCO dataset consists of over 328,000 images of 91 basic object types in naturally occurring contexts. Each image is labeled with at least five captions describing it, which leads to a total of 2.5 million captions, and bounding boxes for each object category that appears in the picture. They also contain 250,000 people annotated with their keypoints.

This dataset has also given rise to image challenges for object detection, keypoints detection and image captioning. The latter is already closed, but there is an open evaluation server and API [Chen et al., 2015] to compare to state-of-the-art methods using several performance metrics such as BLEU, ROUGE, METEOR and CIDEr.

[6] http://www.image-net.org/
[7] *A well-known crowd-sourcing online platform.* https://www.mturk.com/
[8] https://www.kaggle.com/
[9] http://cocodataset.org/#home

PASCAL

PASCAL[10] (*Pattern Analysis Statistical Modeling and Computational Learning*) Visual Object Classes project comprised challenges [Everingham et al., 2010] of image classification, detection and segmentation for eight consecutive years (2005-2012). Nowadays, the challenges are closed but it provides an image database and tools for obtaining these images and their annotations, as well as an evaluation server. The latest dataset (PASCAL 2012 challenge) contained 20 distinct classes and more than 11,000 annotated images. These images were collected from Flickr.

These datasets have annotations for object classification, detection and segmentation, but a subset of the Pascal 2008 challenge dataset was used to create an image description database, called Pascal1K. It contains 1,000 images with objects of different classes and is annotated with five descriptions, generated by humans on Amazon Mechanical Turk crowd-sourcing platform. As it is a medium-scale database, it is generally used as a benchmark for evaluating image description models.

Flickr8K and Flickr30K

Flickr8K[11] [Hodosh et al., 2013] dataset and its extension Flickr30K[12] [Young et al., 2014], contain images from Flickr, each one annotated with five descriptions collected from the crowd-source platform Amazon Mechanical Turk (AMT). Flickr8K consists of approximately 8,000 images and Flickr30K of around 30,000 images. These pictures were collected by the University of Illinois by querying Flickr for specific objects and actions.

SBU

SBU Captioned Photo Dataset was specifically created for the Im2Text image caption generator [Ordonez et al., 2011]. It contains 1 million Flickr images labeled with their original captions generated by their users, collected by querying Flickr for specific objects and actions. It is a large-scale dataset but not very popular for training image description models as the labeled captions may contain subjective information or information not contained in the picture because they were generated by the Flickr user.

CIFAR

CIFAR-10 and CIFAR-100 (*Canadian Institute for Advanced Research*) [Krizhevsky, 2012] are image classification datasets that differ only in the number of classes they have (CIFAR-10 has ten classes and CIFAR-100 one hundred classes, grouped into twenty superclasses).

[10]Download page: http://host.robots.ox.ac.uk/pascal/VOC/
[11]Form for downloading the Filckr8K dataset:https://forms.illinois.edu/sec/1713398
[12]Form for downloading the Filckr30K dataset: https://forms.illinois.edu/sec/229675

Figure 4.4: COCO caption dataset

Both datasets consist of 60,000 32x32 RGB images, CIFAR-10 holds 6,000 in each class and CIFAR-100 holds 600. Keras provides built-in functions to load these datasets.

LabelMe

LabelMe[13] is an image database and an online annotation tool [Russell et al., 2008] created in the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL).

The dataset is dynamic, free to use and open to public contribution, and it consists of almost 200,000 images but not all of them are fully annotated. It has more than 62,000 annotated images and more than 658,000 labeled objects. The online annotation tool allows the user to draw polygons, query images, browse the database or download a subset.

4.4.1 Dataset election

Having studied the characteristics of all these datasets, we decided to use the MS COCO dataset for training and testing our model, as it contains a wide number of images, it is one of the most commonly used for image description and it provides tools for downloading the dataset. They already provide a train/val/test split of the dataset, however, as this dataset was released for their annual challenges, the test set does not come labeled and we will not be able to use it for testing our model, so we will create our own split based on the training and validation sets.

[13]Download page: http://labelme.csail.mit.edu

Figure 4.4 shows some examples of what can be found in COCO caption dataset. In particular, we use the Python COCO API to download the MS COCO dataset and to select a subset of it containing only images of people, to restrict to our objectives the images we provide to the model. In addition, we also use the people's subset (*baby*, *boy*, *girl*, *man* and *woman* categories) of the CIFAR-100 dataset to fine-tune the image feature-extraction model we use, as explained in the next chapter.

Chapter 5

Deep Learning approach for generating image descriptions

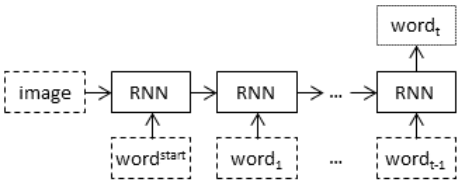
As we have mentioned throughout this project, we have tried to approach the image description generation task using Deep Learning techniques, as they have been proven to be the state-of-the-art in this problem. In this chapter, we describe the model definition and model training processes, as well as all the decisions taken while implementing them.

5.1 Model architecture

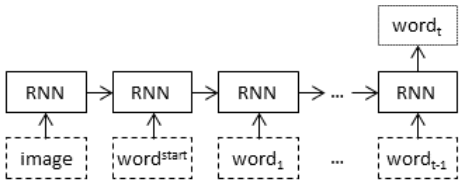
In Chapter 1 we explained that we have restricted ourselves to describing images containing people and that our objective during this project was to determine if this restriction leads to better results when making descriptions about people. For this purpose, we have developed and implemented four different models, all based on the same architecture but each of them focusing more on people's characteristics or, instead, being more general.

Our models' architecture follows a Deep Learning approach based on a Convolutional Neural Network for extracting image features and a LSTM Recurrent Neural Network for the language model generating the descriptions. As we have explained in Chapter 3, they are the state-of-the-art networks in the Computer Vision and Natural Language Generation fields respectively, and it is the structure followed lately in the image description task [Vinyals et al., 2015, Karpathy and Li, 2015, Hendricks et al., 2016].

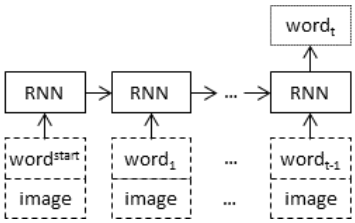
There are different ways that have been explored for combining image and language models. Following *Tanti et al.* nomenclature [Tanti et al., 2017b], this combination can be done by *injecting* image features directly into the RNN that processes the descriptions, so that the language learning process is conditioned by the image features, or it can be done by *merging* the image model with the output of the RNN language model in a later step, so that the learned linguistic characteristics are pure and independent. Moreover, they divide the *inject* architecture in three different types depending on how the image



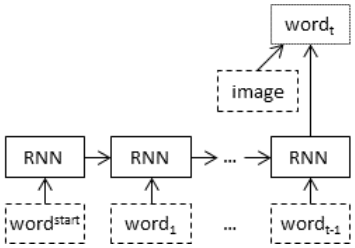
(a) *Init-inject* architecture.



(b) *Pre-inject* architecture.



(c) *Par-inject* architecture.



(d) *Merge* architecture.

Figure 5.1: The different options of combining the image and language models.

[Tanti et al., 2017b]

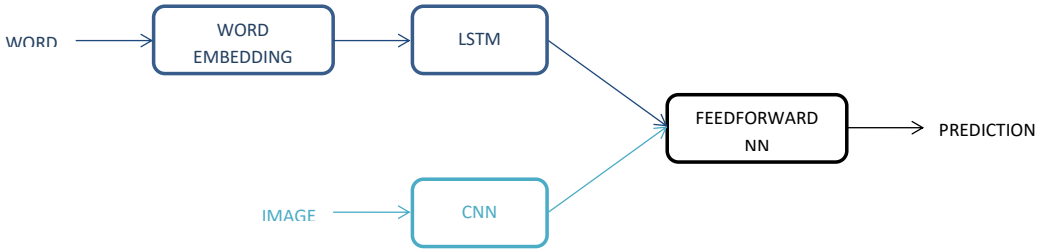


Figure 5.2: Network structure followed by our four models.

features are used: the *init-inject* architecture, that uses them as the initial hidden state of the RNN, the *pre-inject* architecture, that uses them as the first *word* in the description, and the *par-inject*, that uses them as a second input for each step of the RNN. Figure 5.1 shows these four different architectures.

Tanti et al. show that, while *inject* architectures are more commonly used, they are only slightly better at some metrics and the four architectures have similar performance, but *merging* the image features is less memory-consuming, needs less regularization and generates less generic sentences and with richer vocabulary. Therefore, in this project we have followed the *merge* architecture for combining our image and linguistic characteristics.

The structure of our models is shown in Figure 5.2. On the one hand, we have the language generation model, consisting of a word embedding layer that, as we explain later in Section 5.1.2, maps each word into a fixed-length dense vector, and a LSTM layer that learns the linguistic characteristics. We have chosen to use a LSTM layer over a

GRU layer because, as we have mentioned in Section 3.5, it is the one used in almost all image description generators. On the other hand, in the image model, images are fed into a Convolutional Neural Network to extract their features. These two parts are then inputted to a Feedforward Neural Network (following the *merge* architecture) that predicts the next word in the description. Our models follow the same approach as the well-known image description generators and predict at each step the next word in the description based on the description generated so far.

The description of our four models are the following:

- Model 1 is the most general one, it uses a CNN pre-trained on the ImageNet dataset and it is trained and tested on the whole COCO dataset; this model does not make any distinction between images that contain people and images that do not.
- Model 2 *fine-tunes* (re-trains) the pre-trained CNN with a CIFAR-100 subset of only images containing people, to try to focus more on people's characteristics, but it is trained and tested on the whole COCO dataset.
- Model 3 does not fine-tune the pre-trained CNN but it is trained and tested in a subset of the COCO dataset that has only images containing people; this way, image features are general but the language model is focused on people's descriptions.
- Model 4 fine-tunes the pre-trained CNN with the CIFAR-100 subset of only images containing people and its training and testing is restricted to the COCO people subset so that the image and the language models are both focused on people's characteristics and descriptions.

After evaluating the results of these four models we will be able to explain if it is a good approach to restrict to images that contain people in order to obtain better descriptions.

5.1.1 Pre-trained convolutional network

As we have mentioned in Section 4.4, ImageNet is a huge image dataset that also launches annual competitions for image classification and object detection. Every year, the winner teams usually make publicly available their networks and trained weights which are very useful for other image problems, as they provide great results and are trained on a big and general dataset. These weights could be freeze and directly used for prediction or feature extraction as part of a bigger architecture or they could be used as the initial weights of the network and then fine-tuned with another dataset; in both cases, using previously trained networks leads to better results in less computational time than building the Convolutional Neural Network from scratch, making it feasible to train more complex architectures.

Keras provides namesake functions for all the most popular and publicly available models that have been in the top-five leaders of an ImageNet competition, such as Inception, Xception, ResNet or VGG. By calling these functions, Keras downloads into memory their pre-trained weights and returns a Keras `Model` instance of the model.

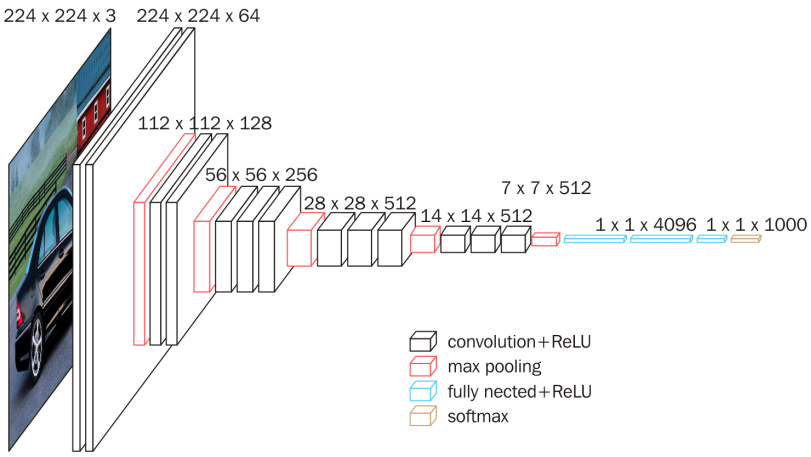


Figure 5.3: VGG16 Convolutional Neural Network architecture.

Following *Vinyals et al.* approach [Vinyals et al., 2015], we have decided to use VGG16 CNN, but we could have used another one obtaining similar results. VGG16 was developed by the Oxford Visual Geometry Group[1] and was the winner of the 2014 ImageNet challenge. It is a very deep convolutional network with small 3x3 filters in all convolutional layers to reduce the number of parameters and max-pooling layers after every two or three convolutional layers. The network's architecture is shown in Figure 5.3.

5.1.2 Word embedding

Typically, natural language algorithms map words into a *1-of-K coding* representation, being K the vocabulary size, that is, each word is encoded into a vector of length K with all the elements being 0 except for a 1 in the position corresponding to the one of that word in the vocabulary. However, this representation builds big sparse vectors if we have a rich vocabulary and similar words lack of any relationship, being completely independent. Word embeddings go beyond this *1-of-K coding* representation building dense vectors where related words have similar representations, reducing the vector length in some orders of magnitude. They are trained in order to learn word relationships and find the mapping from the *1-of-K* vectors into a dense vector space. Therefore, word embeddings are very useful and popular for Deep Learning approaches of Natural Language Processing tasks, raising their performance.

Word embeddings can be learned jointly with the training process of the Deep Learning model using the vocabulary of the dataset of our model, or using already pre-trained weights in a bigger dataset, such as GloVe [Pennington et al., 2014], a word embedding algorithm that has publicly available pre-trained weights in a vocabulary of 400,000 words

[1] http://www.robots.ox.ac.uk/ vgg/

of Wikipedia 2014 and Gigaworld datasets. We use the 100-dimensional GloVe word embedding pre-trained weights for training our models. These weights where downloaded from their webpage[2].

5.2 Fine-tuning VGG16

As we have explained in the previous section, two of the four developed models fine-tune the VGG16 Convolutional Neural Network provided by Keras by retraining it in a subset of images that only contain people of the CIFAR-100 dataset.

Keras provides a function (`keras.datasets.cifar100.load_data()`) for easily downloading CIFAR-100 data. This function downloads the whole dataset. Therefore, to obtain the subset we need, we have selected the indexes where the labels belong to the following set: $\{baby, boy, girl, man, woman\}$ (as it contains all the categories of images with people in them) and we have extracted the subset of these indexes.

Each category in CIFAR-100 contains 600 images, so we have performed data augmentation on the images, with the `ImageDataGenerator` Keras class, to have a bigger dataset to fine-tune VGG16. This class performs data augmentation on the images while progressively generating batches of image data, so that, when used in conjunction with the `model.fit_generator()` function, the whole dataset is not entirely downloaded into memory from the beginning but only when needed, easing memory usage. It allows to perform multiple data augmentation operations, such as random rotations, random zoom or horizontal and vertical flips.

Figure 5.4 shows the piece of the code with the definition of the model that we have implemented for fine-tuning VGG16. We call `VGG16()` Keras function without the top-layer and replace it with Dense layers (fully-connected layers) to flatten the output of the last convolution and a softmax layer to make the predictions. We also make the first layers non-trainable as they have learned common features of every image such as corners or edges, so the already learned parameters work for our task. For re-training this model we use a Stochastic Gradient Descent optimizer (see Section 5.5.1) with a low learning rate (0.001), as we are fine-tuning the model so errors are smaller, and the categorical cross entropy loss, as we are predicting a categorical variable with 5 classes. The categorical cross entropy is an useful loss function to measure how well a model is doing when predicting the output probabilities of a classification problem. Its mathematical expression is:

$$CrossEntropy = -\frac{1}{n}\sum_{x}\sum_{j}[y_j \ln a_j + (1-y_j)\ln(1-a_j)]$$

Where x are the inputs, j the categories, y_j the desired ouput for category j and a_j the actual output for category j. We have trained this model using `model.fit_generator()`

[2]https://nlp.stanford.edu/projects/glove/

```
"""-----Definition of VGG16 fine-tuned model-----"""
def model_definition():
    vgg16_model = VGG16(weights="imagenet",
                        include_top=False,
                        input_shape=(224,224,3))

    # Replace top layer
    vgg16_output = vgg16_model.output
    x1 = Flatten()(vgg16_output)
    x2 = Dense(4096, activation='relu', name="fc1")(x1)
    x3 = Dense(4096, activation='relu', name="fc2")(x2)
    prediction = Dense(100, activation='softmax', name="predictions")(x3)

    # Create model
    model = Model(input=vgg16_model.input, output=prediction)

    # Set first layers non-trainable
    for layer in model.layers[:12]:
        layer.trainable = False

    model.compile(optimizer=SGD(lr=1e-3, decay=1e-6, momentum=0.9),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Figure 5.4: Python code of VGG16 fine-tuned model definition.

function with a batch size of 32 images and 30 epochs[3], and save it in a hdf5[4] file into memory using `model.save()`.

5.3 Data preparation

Once we have our fine-tuned VGG16 model, we can start defining the models for generating image descriptions. As we explained in the first section of this chapter, we will use COCO dataset, partly or entirely depending on the model, for training and testing our models. Fortunately, COCO provides a Python API for loading into memory and parsing the dataset in an easier way. To use this API, the dataset files must be downloaded from their webpage in an specific directory structure. We have downloaded the 2014 dataset because it is the one that contains image descriptions as labels. After cloning the github project and running the Makefile, the API can be used by importing the `pycocotools.coco`

[3] An *epoch* is an entire iteration over the whole training dataset. The *batch size* is the number of training samples trained between parameters updates.

[4] File format to store large amounts of data.

module. More information on how to use this API can be found on their webpage[5] and their github project[6].

All COCO images and descriptions must be preprocessed in order to correctly use them as the inputs of our models. Even though two of our models work only with a subset of COCO, the other two work with the whole dataset, so both image preprocessing and descriptions preprocessing have been done in the whole dataset, leaving the subset extraction to a later step where we also perform the train/val/test split of the dataset as we explained in Section 4.4. In the following subsections we explain how we have carried out these two tasks.

5.3.1 Image preprocessing

COCO images do not need too much preprocessing. However, the Convolutional Neural Network of our models, either the VGG16 network or the fine-tuned model, has already precomputed its weights and these weights will remain frozen when training our image descriptions generation models. Therefore, images features could be precomputed before training the models and saved to be used later as an input instead of the images themselves. This way we only calculate these features once instead of every time the training process passes through one of these images, saving computational time and memory space.

This image preprocessing must be repeated for the VGG16 model and for our fine-tuned model, to extract the features obtained by both models and use one or another depending on which of the four models we are training. Figure 5.5 shows the features extraction function we have defined, where `model` refers to either `VGG16()` or to our fine-tuned model, `features` is the dictionary that will store the extracted image features and `images_ids` are all COCO training image ids obtained with COCO API as follows:

```
dataDir = ".."
dataType = "train2014"
annFile = '{}/annotations/instances_{}.json'.format(dataDir,dataType)
coco = pycocotools.coco.COCO(annFile)
imgs_ids = coco.getImgIds()
```

This process is also done with the validation COCO dataset, changing `dataType` to `"val2014"`. To extract image features, last layer of the model must be removed using `layers.pop()` because this layer is the one that predicts which class the image belongs to and we only care about image features. We then load every image in the COCO dataset using COCO API and Keras `load_img()` function. Image target size is 224x224 as this is the input size of VGG16 (and therefore also of our fine-tuned model). Once we have loaded the image into memory, it is converted to a numpy array and reshaped, using Keras functions from the `keras.preprocessing.image` module. Finally, features are extracted

[5] http://cocodataset.org/download
[6] https://github.com/cocodataset/cocoapi

```
def features_extraction(model, features, directory, dataType, images_ids):
# remove prediction layer from model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)

    for id in images_ids:
        file = "0:0>12".format(id)
        path = "/images//COCO__.jpg".format(directory, dataType, dataType, file)
        img = load_img(path, target_size=(224, 224))

        # img to 3D numpy array (height, weight, color channel) and reshape it
        img = img_to_array(img)
        img = img.reshape((1, img.shape[0], img.shape[1], img.shape[2]))

        # get features and store them in dict
        img = preprocess_input(img)
        feature = model.predict(img, verbose=1)
        features[id] = feature

    return features
```

Figure 5.5: Image preprocessing: features extraction of COCO images.

using the `predict()` function and then stored into the dictionary. After having extracted all training and validation images features, we save the dictionary containing them in a file, using Python `pickle` library[7], for later use.

5.3.2 Text preprocessing

The image descriptions preprocessing task that we have implemented performs various operations on the words to clean those descriptions and obtain a suitable vocabulary for learning, that should be representative and expressive enough. These operations are: changing uppercase letters to lowercase, removing punctuation marks, removing numbers, removing words of length one (as they are not expressive), removing words used less than five times in the whole dataset and transforming English contractions such as *don't* to *do not*. We also add initial and final tokens (*START* and *END*) for delimiting the descriptions.

Figure 5.6 shows the preprocessing function we have defined for cleaning COCO descriptions, where: `word_freq` is a Python Counter[8] with the words' frequencies and `descriptions` is a list containing all the descriptions of COCO training dataset and it is obtained using COCO API as follows:

[7] https://docs.python.org/3/library/pickle.html
[8] https://docs.python.org/3/library/collections.html#collections.Counter

```
def preprocess_desc(descriptions, word_freq):
    # translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)

    for desc_info in descriptions:
        desc = desc_info['caption']
        words = desc.split()
        # convert to lower case
        words = [w.lower() for w in words]
        # remove numbers
        words = [w for w in words if not w.isnumeric()]
        # transform contractions
        words = [contractions(w) for w in words]
        # remove punctuation from each word
        words = [w.translate(table) for w in words]
        # remove words length < 2
        words = [w for w in words if len(w)>1]
        # remove words used < 5 times
        words = [w for w in word_freq.keys() if word_freq[w]>=5]
        # store in dictionary with START and END tokens
        desc_info['caption'] =  'START ' + ' '.join(words) + ' END'
```

Figure 5.6: Text preprocessing: cleaning COCO descriptions.

```
dataDir = ".."
dataType = "train2014"
annFile = '/annotations/captions_.json'.format(dataDir,dataType)
coco = COCO(annFile)
anns_ids = coco.getAnnIds()
descriptions = coco.loadAnns(anns_ids)
```

This process is also repeated with the validation descriptions, as we have done with image preprocessing, changing `dataType` to `"val2014"`. Each element of the `descriptions` list is a dictionary containing the image id the description belongs to, an id for the description and the description itself. For transforming contractions, we have defined a dictionary containing the transformations and `contractions()` returns this transformation if the word is in that dictionary. Once we have preprocessed all training and validation descriptions, we merge both lists into a single one and save it in a file using `pickle` library.

5.4 Dataset split

After having preprocessed the entire COCO dataset, we now split it into train, validation and test sets (with a 70%-20%-10% division) and divide consequently the features dictionaries (VGG16 features and fine-tuned model features) obtained while preprocessing the

```
# get ids of 35000 training imgs general and with people
dataDir = ".."
dataType = "train2014"
annFile = '/annotations/instances_.json'.format(dataDir,dataType)
coco = COCO(annFile)
cats_ids = coco.getCatIds(catNms=['person'])
train_people_ids = coco.getImgIds(catIds=cats_ids)
train_people_ids = random.sample(train_people_ids, 35000)

train_ids = coco.getImgIds()
train_imgs_ids = random.sample(train_ids, 35000)

# get ids of 35000 validation imgs general and with people
dataDir = ".."
dataType = "val2014"
annFile = '/annotations/instances_.json'.format(dataDir,dataType)
coco = COCO(annFile)
cats_ids = coco.getCatIds(catNms=['person'])
val_people_ids = coco.getImgIds(catIds=cats_ids)
val_people_ids = random.sample(val_people_ids, 15000)

val_ids = coco.getImgIds()
val_imgs_ids = random.sample(val_ids, 10000)

# split val people in val and test
val_people_ids, test_ids = train_test_split(val_people_ids, test_size=5000)
```

Figure 5.7: Test split of COCO dataset.

images and the descriptions list obtained while preprocessing the descriptions.

As our goal is to evaluate how well our models describe images containing people, the test set must only include this kind of images. On the other hand, the train and validation sets, may or may not include other kind of images, depending on the model that we are training. Therefore, we will have two different splits for these two sets, one of them of the subset of images containing people (excluding the ones in the test set) and the other one of the whole dataset without the test set.

Training our models with the whole COCO dataset (more than 300,000 images) was impossible due to the characteristics of the server we have used and to our time limitation. In order to reduce time and memory consumption, we decided to restrict our training process to only a subset of the COCO dataset: 35,000 images for the training set, 10,000 images for validation and 5,000 images for the test set (70%-20%-10%).

To obtain the training set we have randomly selected, using `random.sample()` Python function, 35,000 images of the whole COCO training set and 35,000 images of all the COCO training images that contain people. Validation and test sets have been extracted both

from COCO validation set, randomly selecting 15,000 COCO validation images containing people (5,000 for the test set and the remaining 10,000 for the validation set) and 10,000 COCO validation images of any kind. This process is shown in Figure 5.7. We have also ensure that the general training and validation splits are representative enough by verifying that the random subsets have enough images that contain people (30%-60%).

Once we have performed the dataset splits, we divide the features dictionaries and the descriptions list accordingly to the different splits and we save these divisions in memory using `pickle` library.

5.5 Training process

The first step in the training process of the four models is to load the files obtained with the image and text preprocessing and divided when making the dataset split. Following the enumeration we have defined in Section 5.1, Model 1 and Model 3 load image features obtained with VGG16 and Model 2 and Model 4 load the ones obtained with the fine-tuned VGG16 model. Subsequently, Model 1 and Model 2 load the train and validation split of the whole dataset and Model 3 and Model 4 the one of the subset of images that contain people.

The remaining steps in the training process are the same for the four models, with the only differences being the inputs. After loading the images features and descriptions, we create and fit a Tokenizer over all training and validation descriptions, using Keras `Tokenizer()` class. A Tokenizer learns from a list of texts how to map the bag of words contained in that texts to integer values and allowing to *see* each description as an integer sequence.

The next step is to define the model that we are going to train. The model definition in Python using Keras follows the architecture that we have defined in Figure 5.2 and it is shown in Figure 5.8. The hyperparameters are extracted from *Tanti et al.* observations [Tanti et al., 2017b]. `vocab_size`, `max_length` and `emb_matrix` arguments are the size of the whole bag of words contained in the descriptions, the length of the largest description and the GloVe word embedding weight matrix respectively.

The function starts with the definition of the Keras input tensor for the image features using `Input()` and the input shape is 4096 as it is the shape of the features vector extracted for each image. This input is passed to a Dense layer with a ReLu activation function and L2 regularization to obtain a 128 element representation. Regularization techniques are introduced in Deep Learning training processes to prevent or reduce overfitting (good learning of the traning set but too specific, leading to no generalization power). In particular, L2 regularization adds an extra term to the cost function proportional to the sum of squares of all the weights. We then define the language model with a Keras `Input()` tensor of length `max_length` and this input is passed to a word embedding layer with GloVe pre-trained weights to map the sparse word vectors to dense ones, as we explained in Section 5.1.2. After the Embedding layer we perform a 0.5 dropout [Srivastava et al., 2014],

```
def model_definition(vocab_size, max_length, emb_matrix):
    # feature extractor model
    img_inputs = Input(shape=(4096,))
    fe1 = Dense(128, kernel_regularizer=l2(1e-8), activation='relu',
                name="features")(img_inputs)

    # description model
    desc_inputs = Input(shape=(max_length,))
    de1 = Embedding(vocab_size, 100, weights=[emb_matrix],
                    trainable=False)(desc_inputs)
    de2 = Dropout(0.5, name="text_dropout")(de1)
    de3 = LSTM(128, name="LSTM_layer")(de2)

    # merge model
    merge_inputs = [fe1,de3]
    me1 = add(merge_inputs)
    me2 = Dense(128, activation='relu', name="decoder")(me1)
    outputs = Dense(vocab_size, kernel_regularizer=l2(1e-8),
                    activation='softmax', name="predictions")(me2)

    model = Model(inputs=[img_inputs, desc_inputs], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    model.summary()

    return model
```

Figure 5.8: Image description generation model definition with Keras.

a regularization technique used to prevent overfitting that consists in ignoring while training some neurons selected randomly. This is followed by a Keras LSTM layer with 128 memory units. Next we merge both the output of the LSTM layer and the output of the Dense layer and apply a Dense layer with ReLU activation function and a softmax Dense layer, with L2 regularization, over the vocabulary size to make the prediction of the next word. This model is trained with a categorical cross entropy loss (as the output of the network are the probabilities of the words in the vocabulary to be the next word in the description) and Adam optimizer (see Section 5.5.1) with the default values, as *Tanti et al.* suggest [Tanti et al., 2017b].

Finally, the last step is to fit the model with the training and validation data that we have loaded. As COCO is a very big dataset and to ease memory usage, we use a generator to produce training and validation data and fit the model with the `fit_generator` function. To control the training process, we use the `fit_generator` function with a `ModelCheckpoint` callback, to save the model after each epoch monitoring the loss on the validation set, and a `TensorBoard` callback, to save tensorboard logs for visualization (as we explained in Section 4.3).

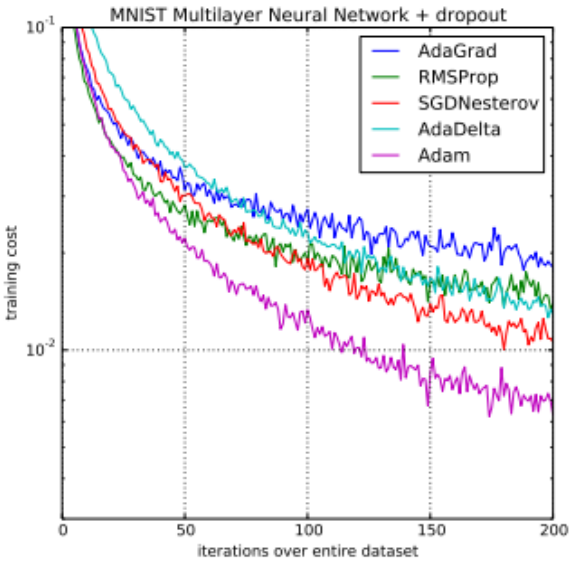


Figure 5.9: Optimization algorithms steps reaching local optimum.

We have trained the four models 35 epochs in the server with the GPU. The whole process (VGG16 fine-tune, data preparation and training of the four models) lasted around eight days.

5.5.1 Optimizers

As we explained in Section 3.4, Deep Learning models learn by trying to find the optimal parameters that minimize the cost function and the gradient descent method is an optimization algorithm to approximate a local optimum by computing the gradient of the function. However, the cost function depends on all training samples, and therefore so does the gradient, which makes it too expensive to compute for Deep Learning models that are typically trained on a big dataset. For this reason, as the cost function is usually a sum over the training set, an incremental approach for calculating the gradient called Stochastic Gradient Descent (SGD) was developed. It calculates the gradient based only on a small subset of the training examples to speed up this calculation. This usually involves taking more iterations to approximate the minimum, but faster.

In SGD, weights are updated using the same learning rate for all weights and all iterations, making it difficult or slow to converge to the minimum if the learning rate is too big or to small. Many variants have been proposed and developed in order to update this SGD algorithm to try to reach faster the local minimum.

Some of this variants are AdaGrad [Duchi et al., 2011], Adam [Kingma and Ba, 2014] and RMSProp. AdaGrad maintains a per-parameter learning rate instead of having the

same learning rate for all the parameters, updating it based on the sum of previous gradient values. So does RMSProp, that also has a learning rate for each parameter but these learning rates are updated based on the average of recent gradient calculations, that is, based on how quickly the gradient is changing. Adam is an improvement of RMSProp that updates the learning rates based on the average but also on the gradients variance, this is the one we use for training our model. Figure 5.9, obtained from Google Images, shows how quickly these algorithms reduce the cost function.

As mentioned before, our models have been trained using the Adam optimizer.

Chapter 6

Evaluation and results

Once we have our four models defined and trained, we can evaluate them on the test set and we can use them to generate descriptions of new unseen images. In this chapter, we explain how we have tested the models, we define the metrics that we have used and we show the results obtained, including some examples of images and their generated descriptions.

In Section 5.4 we explained that, as our objective is to determine how well our models perform on describing images that contain people, the division of the dataset was made so that the test set has only this kind of images. Therefore, the results of the evaluation process and the examples that we show in this chapter refer all to images containing people.

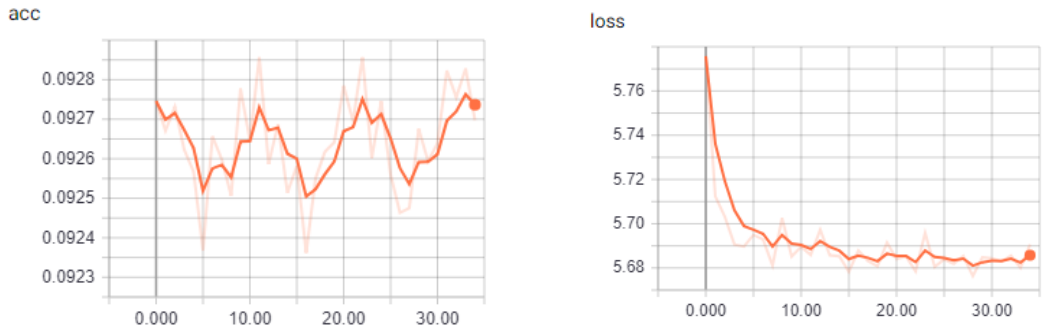
6.1 TensorBoard visualization

As described in Section 5.1, the four models that we have developed follow the same architecture but differ on the techniques and datasets used. The characteristics of these four models are the following:

- Model 1: the most general one. Uses a pre-trained VGG-16 CNN and it is trained on MS COCO dataset.
- Model 2: uses a fine-tuned version of VGG-16 and it is trained on MS COCO dataset.
- Model 3: uses a pre-trained VGG-16 CNN and it is trained on a subset of MS COCO dataset of images containing people.
- Model 4: uses a fine-tuned version of VGG-16 and it is trained on a subset of MS COCO dataset of images containing people.

(a) Accuracy (b) Loss

Figure 6.1: Model 1 TensorBoard charts.



(a) Accuracy (b) Loss

Figure 6.2: Model 2 TensorBoard charts.

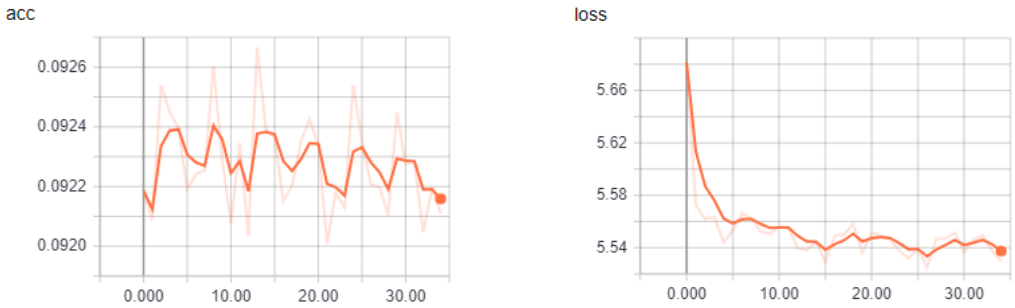
In Section 4.3 we explained that TensorBoard is a suite of visualization tools developed by Tensorflow that plots metrics' evolution over the training process. Figures 6.1, 6.2, 6.3 and 6.4 show accuracy and loss charts of Model 1, Model 2, Model 3 and Model 4 respectively. These charts are obtained from TensorBoard log files generated during the training process.

Analyzing these charts we see that Model 1 and Model 3 both have an increase in their accuracies and a decrease in their losses, which is a good thing that shows that the models are learning correctly. However, the evolution of Model 2 and Model 4 accuracies is maintained almost constant and their losses decrease less and slower.

We can then affirm that accordingly to these charts, Model 1 and Model 3 are better models than Model 2 and Model 4 and that they may perform better generating images descriptions. In the next section we will remark if the evaluation process on the test set confirms this assumption.

(a) Accuracy (b) Loss

Figure 6.3: Model 3 TensorBoard charts.



(a) Accuracy (b) Loss

Figure 6.4: Model 4 TensorBoard charts.

6.2 Quantitative analysis

The image and text preprocessing tasks, as we illustrated in Section 5.3, were done on the complete dataset and the train/val/test division was made afterwards. Therefore, we already have the precomputed test images features and their preprocessed descriptions, so for testing the models that we have developed, first of all, we need to load those images features and preprocessed descriptions with `pickle` library, as well as the hdf5 models' files using Keras `load_model` function.

Once we have all loaded, for each model we need to generate the descriptions it predicts for every image in the test set and we need to save them all together in a list. We repeat this process for the four models in order to obtain the different descriptions that each one predicts.

As Figure 6.5 shows, with all the generated descriptions and their corresponding *true* descriptions (the ones provided with the dataset) for every image in the test set, we compare the results using the most well-known metrics for Natural Language Generation. `metrics.score()` function is used to obtain the output of each metric. These metrics and

```
def evaluate_model(model, model_name, features, tokenizer, max_length, descs):
    true_descs = {}
    predicted_descs = {}

    # generate all descriptions
    for key, desc_list in descs.items():
        pred_desc = generate_description(model, features[int(key)],
                                         tokenizer, max_length)
        true_descs[key] = [" ".join(desc) for desc in desc_list]
        predicted_descs[key] = [pred_desc]

    # compute metrics values
    scores = metrics.score(true_descs, predicted_descs)
```

Figure 6.5: Evaluating the performance of a given model.

the results obtained are later explained in Section 6.2.1.

To generate a description for a given image, we use a greedy approach that builds the sentence by selecting at each step the word predicted with the highest probability. Figure 6.6 shows the function that we have defined for generating descriptions using this approach. The arguments of this function are: `model`, it is the model we are going to use to generate the description, `features`, the image features extracted with VGG16 or its fine-tuned version, `tokenizer`, it is the tokenizer defined while training the model, and `max_length`, the maximum length permitted for a description.

To start the image description generation process we must input the `START` token by converting it to an integer using the tokenizer `texts_to_sequences()` method and padding it to the maximum length that each model has defined for a sentence. Once we have the first input of the model, we iteratively predict the next word in the sentence using Keras `model.predict()` function and selecting the one with highest probability, until we predict the `END` token or until we reach the maximum length.

6.2.1 Metrics

There are several metrics that have been defined to evaluate how well NLP models perform. Our goal is to use these metrics to compare our four models with each other and also to compare them with state-of-the-art results. For this reason, we have selected the most well-known and widely used ones for machine translation and image description generation so that we can make those comparisons. These metrics are: BLEU metrics (BLEU-1, BLEU-2, BLEU-3 and BLEU-4), ROUGE-L, METEOR and CIDEr.

BLEU (*Bilingual Evaluation Understudy*) metrics [Papineni et al., 2002] were developed to evaluate machine translation models and they are now the most popular metrics for many NLG problems (text summarization, image description generation, speech recog-

```
def generate_description(model, features, tokenizer, max_length):
    input_desc = "START"

    for i in range(max_length):
        # tokenize and pad input
        tok_desc = tokenizer.texts_to_sequences([input_desc])[0]
        pad_desc = pad_sequences([tok_desc], maxlen = max_length)

        # next word prediction
        pred = model.predict([features,pad_desc], verbose=0)
        max_prob = np.argmax(pred)
        word = get_word(max_prob, tokenizer)

        # finish if no possible mapping
        if word is None:
            break

        # finish if END token
        if word == "end":
            input_desc = input_desc + " END"
            break

        # append word to input_desc
        input_desc = input_desc + " " + word

    return input_desc
```

Figure 6.6: Generation of the predicted description for a given image.

nition...). BLEU-1, BLEU-2, BLEU-3 and BLEU-4 metrics compute individually for m from 1 to n, with $n \in \{1, 2, 3, 4\}$ respectively, the number of m-grams in the predicted text that are present in any of the reference texts and then its weighted geometric mean.

ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*) [Lin, 2004] is a set of metrics commonly used for text summarization evaluation, but it is also used to evaluate other NLG systems. ROUGE-L is based on the Longest Common Subsequence, that is, the longest subsentence that is present in the predicted and the reference sentence.

METEOR [Denkowski and Lavie, 2014] (*Metric for Evaluation for Translation with Explicit Ordering*) metric for NLG problems is more recent and it is computed by trying to generate a 1:1 correspondence between the words in the predicted and reference sentences.

CIDEr (*Consensus-based Image Description Evaluation*) [Vedantam et al., 2014] is also a rather new metric and it is specific for evaluating image description generators. CIDEr is calculated based on the TF-IDF[1] weighting of each n-gram.

[1]Term Frequency Inverse Document Frequency.

	Bleu-1	Bleu-2	Bleu-3	Bleu-4	ROUGE-L	METEOR	CIDEr
Model 1	0.574	0.355	0.257	0.104	0.392	0.158	0.341
Model 2	0.014	8.7e-12	1.1e-10	4.1e-10	0.039	0.006	0.0
Model 3	0.602	0.389	0.272	0.124	0.407	0.172	0.364
Model 4	0.031	9.2e-12	7.1e-15	2e-16	0.105	0.008	2.9e-13
State-of-the-art	0.617	0.419	0.285	0.191	0.422	0.175	0.378

Table 6.1: Metrics results.

To compute the score of each metric for the four models developed, we have used `metrics.score()` function from the `metrics` library obtained from Github[2]. This library uses COCO evaluation server code[3] for the score calculation of each metric.

The results obtained for each model from the testing process are shown in Table 6.1. This table also shows the state-of-the-art scores [Tanti et al., 2017a] for the computed metrics. However, whereas our results are obtained from evaluating only on images containing people, state-of-the-art results come from any kind of images. With these results we can confirm what we predicted in Section 6.1: Model 1 and Model 3 perform better than Model 2 and Model 4; what is more, Model 2 and Model 4 have very bad results.

As we defined in Section 5.1, Model 2 and Model 4 were trained on a fine-tuned version of the CNN used. Therefore, the fact that they have such bad results leads us to conclude that it was not a good idea to fine-tune it. This could be produced because the CNN may have overfitted when fine-tuned, or because it was fine-tuned on CIFAR-100, the models were trained on COCO dataset and these two datasets might have some differences.

Nevertheless, it can be seen in Table 6.1 that, in general, Model 3 performs slightly better than Model 1, being Model 3 the best one. Therefore, we can state that it was a good practice to train our models on a subset of COCO dataset with only images containing people, restricting ourselves and focusing our efforts on those kind of images. That is, we were right, at the beginning of the project, believing that people are described in a different manner.

Finally, comparing in Table 6.1 the scores obtained for Model 1 and Model 3 to state-of-the-art results, we can see that they perform similarly even though we made some restrictions during our training, so we are satisfied with the models we have developed.

6.3 Qualitative analysis

In this section we show some examples of images containing people and the descriptions generated by our models. To generate descriptions of unseen images, first of all, we must use the Convolutional Neural Network, either VGG-16 of its fine-tuned version, to extract

[2]https://github.com/kelvinxu/arctic-captions
[3]https://github.com/tylin/coco-caption

the image features. Then we have to load the different models and their corresponding tokenizers and use the function described in Figure 6.6 to obtain the descriptions.

We have generated Model 2 and Model 4 descriptions for some examples and for every image, the output is always the same: Model 2 always outputs *'start start start start...'* and Model 4 *'start end'*. This explains the results obtained in Table 6.1 and, as we imagined in the previous section, the reason might be the differences between the dataset used for finetuning the CNN (CIFAR-100) and the one used for training the network (COCO dataset). Therefore, we show in this section the descriptions that Model 1 and Model 3 generate.

Figures 6.7, 6.8, 6.9 and 6.10 show examples of good descriptions generated by Model 1 and Model 3. Figures 6.11, 6.12 and 6.13 show examples where these two models have different perspectives of the image: Model 3 has been trained to focus more on people while Model 1 is more general. Bad image descriptions generated by these models are shown in Figure 6.14 and Figure 6.15.

Analyzing these examples we can conclude that Model 1 and Model 3 can produce good descriptions of the input images and that, even though they can sometimes fail describing what there is in an image, they generate well-formed sentences. We can also confirm with these examples the results obtained in Table 6.1.

Model 1: tennis player is holding racket
Model 3: tennis player is swinging tennis racket

Figure 6.7: Good descriptions.

Model 1: person in the snow holding snowboard down snow
Model 3: man on skis in the snow

Figure 6.8: Good descriptions.

Model 1: woman on beach with surfboard
Model 3: two woman riding surfboard on the beach

Figure 6.9: Good descriptions.

Model 1: man in baseball uniform is holding bat
Model 3: baseball player is swinging bat at baseball game

Figure 6.10: Good descriptions.

Model 1: bird flying on beach
Model 3: man is riding surfboard on the beach

Figure 6.11: Different perspectives.

Model 1: pizza with cheese and cheese on it
Model 3: person is sitting at table with pizza

Figure 6.12: Different perspectives.

Model 1: plate of food with meat and vegetables on it
Model 3: person is sitting on table with some food

Figure 6.13: Different perspectives.

Model 1: person in the snow on skis on the snow
Model 3: man riding snowboard down snow covered slope

Figure 6.14: Bad descriptions.

Model 1: man holding hot dog in front of pizza
Model 3: man is sitting at table with pizza

Figure 6.15: Bad descriptions.

Chapter 7

Conclusions and future work

7.1 Conclusions

We conclude this project having accomplished the main objectives established at the beginning. We have studied and learned different Deep Learning techniques and tools, that are nowadays very important in the Artificial Intelligence field. We have also reviewed all the bibliography regarding the image description generation problem.

We have carried out this project in a server with a GPU provided by GAIA research group, as training Deep Learning models needs a lot of computing capacity. We have used Keras on top of Tensorflow as the Deep Learning framework and TensorBoard for generating plots of the training process. We have also analyzed the different available and well-known image datasets in order to select the ones that were more suitable for our project.

Finally, we have achieved to develop four different Deep Learning models to automatically generate descriptions of images containing people. Our four models were all built using state-of-the-art Deep Learning techniques for Computer Vision and for Natural Language Generation: Convolutional Neural Networks for extracting the image features and Recurrent Neural Networks for generating the sentence, as well as Feedforward Neural Networks for building the models' outputs. These models were successfully trained and tested on a big and popular image dataset.

To evaluate our models we have analyzed the TensorBoard plots generated from the training data, we have performed a quantitative analysis using the most popular metrics and a qualitative analysis generating descriptions for unseen images in order to visualize and confirm the results obtained from the quantitative analysis.

These evaluation analyses lead all to the same conclusions: we obtained desirable results from the evaluation process, similar to state-of-the-art results, for two of the four models developed (Model 1 and Model 3). However, fine-tuning was not a good practice, as Model 2 and Model 4 (the ones that fine-tuned the CNN) have poor evaluation results,

maybe because of the differences between the datasets used for fine-tuning and training. Nevertheless, Model 3 have slightly better results than Model 1 so we can confirm that our thoughts were in the right path at the beginning of the project: images containing people are described differently and so it is a good practice to restrict the training process to this kind of images.

7.2 Future work

Considering concluded the main objectives of this project, there are many directions in which future work could go in order to broaden it. First of all, data quantity and quality are essential for a good training, and hence the model could be more accurate and have better evaluation results with more, and more representative, examples in the dataset and labeled more precisely.

Another option is to improve our hardware capacity by using more GPUs, a better one or by another way of obtaining more computational power in order to being able to train the models more rapidly.

In addition to all of these changes to accomplish better results with our models, another direction we could take in the future is to broaden the scope of the model's objective. We could include more complete image descriptions depicting more complex characteristics of the person in the image, such as the person's feelings (e.g. if it is smiling or sad), or other characteristics that are not explicitly shown in the picture but that can be derived from it, not limiting ourselves to describing what it is strictly in the picture. Moreover, the model could be extended to the description of, not only people, but other objects appearing in the picture, as well as the spatial relationship between those objects and the people (e.g. *the man behind the window*, *the woman next to the blue chair*).

There are also some emerging tasks that combine both computer vision and natural language fields, trying to go beyond what it is strictly depicted in the image, that could be interesting to study in a future work of this project; for example, the task of VQA (Visual Question Answering) [Wu et al., 2017a], that tries to find an answer to a given question about a given image, or visual storytelling [Huang et al., 2016, Mostafazadeh et al., 2017], that generates a narrative description from the input image by making subjective assumptions of what it is happening.

Another interesting possibility is to implement a machine learning model, using state-of-the-art machine learning algorithms such as xgboost, in order to analyse the differences between both deep learning and machine learning models (computing time, dataset needs, accuracy, etc).

Finally, although Convolutional Neural Networks are the state-of-the-art, they have some limitations that decrease the level of accuracy of the model. In the last months, a new kind of neural networks trying to solve these limitations, called Capsule Neural Networks, have arisen. It would be a good future direction of this project to implement

the solution using this kind of network and to compare it to the convolutional one to see if it yields better results.

7.2.1 CNN limitations

Convolutional Neural Networks are until now the state-of-the-art approach in the image analysis field, but they have some limitations when it comes to spatial relationships.

Convolutional Neural Networks became a major advantage compared to traditional feedforward networks in the image field, as they allowed a 2-D matrix as the input of the model in contrast with the need of flattening the pixel matrix into a vector, which, along with the convolution and pooling operations, permitted the model to be invariance to translations.

As we have already discussed during this project, the job of the convolutional layers in a CNN is to detect important features in the image, with layers closer to the input detecting simple features and deeper layers close to the output combining them to detect more complex features. On the other hand, pooling layers, being max pooling the most widely used, help reducing the image dimensionality, and therefore computational time, and summarizing the important information present in the image, as well as creating spatial invariance.

However, this invariance created by the max pooling layers loses the existing spatial relationships between all these features. It does not take into account how the different features are related to each other, since max pooling loses their precise locations. This makes the model output a false positive when the image has the components of an object but not in the correct order. For example, considering it is a face when it has two eyes, a nose and a mouth, but the mouth is where an eye should be and vice versa.

Furthermore, CNNs do not take into account the spatial characteristics of each feature. That is, they do not recognise an object they have already seen, shown now in a different orientation. Continuing with the example above, an image with a face turned upside down is not detected as a face by the CNN. To combat this, convolutional models are trained by giving images of different possible angles explicitly to the network.

Nevertheless, the solution is not to remove the max pooling layers from the model, as we really need to introduce some kind of invariance; otherwise, the model will only recognise images very similar to the ones in the training set. What a better model needs is to adjust this invariance with *equivariance*, that is, understanding rotation and proportion changes and adapting accordingly.

7.2.2 Capsule neural networks

Geoffrey E. Hinton, known as the *father* of Deep Learning, has been discussing about convolutional networks' limitations and the need to suggest new models to cover them, for

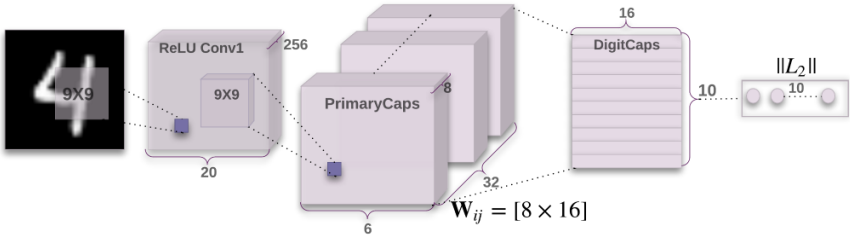


Figure 7.1: Capsule neural network structure (CapsNet).

a long time now. In 2011 he published a paper [Hinton et al., 2011] regarding a proposal of including more complex spatial characteristics to neural networks, but it has not been until recently that he has finally published two papers [Hinton et al., 2018, Sabour et al., 2017] explaining his new sophisticated neural network model, Capsule Networks.

In the paper they have published [Sabour et al., 2017], they propose CapsNet, a capsule neural network model for the MNIST[1] dataset, reporting better results than CNNs and without the need of data augmentation. This model is represented in Figure 7.1.

Hinton believes that it is necessary to preserve pose's (translation and rotation) hierarchy between features. This way it is easier for a model to recognise another perspective of something it has already seen. These neural networks try to overcome those Convolutional Networks' limitations by explicitly taking into account those spatial relationships between features, whereas CNNs do not have this 3-D space understanding.

This is achieved by replacing the scalar outputs of traditional neurons with vector outputs encoding the features' locations and changing max-pooling with a dynamic routing algorithm. Thus, an object presence is derived from not only the presence of its constituent parts, but also from them being at the right locations.

According to Hinton, *A capsule is a group of neurons whose outputs represent different properties of the same entity.* They are nested layers within a layer, each one focusing on detecting a particular feature in the image and outputting a vector representing the feature's existence and its pose properties.

The length of the vector represents the probability of the existence of the feature and the spatial properties are encoded in the vector's direction. This way, when the detected feature moves or changes its spatial state, the length of the vector does not change (the probability remains the same) but it changes its orientation.

In these neural networks, capsules in a lower layer decide dynamically how to send its output vector to the next layer's capsules. Each capsule in the lower layer computes, for each possible parent, a prediction vector of the pose of a higher-level capsule feature

[1]Dataset of handwritten digits.

(what the higher capsule would *see*), by multiplying its output by a weight matrix. When several capsules in one layer agree on what they may have detected, they activate the corresponding capsule at the next layer.

Hinton calls this method the *routing-by-agreement algorithm*, and it substitutes the max-pooling algorithm.

Chapter 8

Conclusiones y trabajo a futuro

8.1 Conclusiones

El trabajo concluye habiendo cumplido los principales objetivos establecidos al comienzo de este. Hemos estudiado y aprendido diferentes técnicas y herramientas de aprendizaje profundo, que son hoy tan importantes en el campo de la Inteligencia Artificial. También hemos revisado toda la bibliografía relativa a la generación de descripciones de imágenes.

Hemos llevado a cabo este proyecto en un servidor con una GPU proporcionado por el grupo de investigación GAIA, ya que entrenar modelos de aprendizaje profundo necesita mucha capacidad computacional. Hemos usado Keras sobre Tensorflow como entorno de trabajo de aprendizaje profundo y TensorBoard para generar gráficas del proceso de entrenamiento. También hemos analizado los distintos *datasets* disponibles para seleccionar los que eran más adecuados para este proyecto.

Por último, hemos conseguido desarrollar diferentes modelos de aprendizaje profundo para generar automáticamente descripciones de imágenes que contienen personas. Nuestros cuatro modelos se han construido todos usando técnicas del estado del arte de aprendizaje profundo para visión por computador y para generación del lenguaje natural: redes neuronales convolucionales para extraer las características de la imagen y redes neuronales recurrentes para generar la frase, así como redes neuronales *feedforward* para construir el output de los modelos. Estos modelos se han entrenado y evaluado satisfactoriamente en un *dataset* de imágenes grande y conocido.

Para evaluar nuestros modelos hemos analizado las gráficas de TensorBoard generadas de los datos del entrenamiento, hemos realizado un análisis cuantitativo usando las métricas más conocidas y hemos realizado un análisis cualitativo generando descripciones de nuevas imágenes para visualizar y confirmar los resultados obtenidos del análisis cuantitativo.

Estos análisis de evaluación llevan todos a las mismas conclusiones: hemos obtenido del proceso de evaluación resultados deseables, similares a los resultados del estado del arte,

para dos de los cuatro modelos desarrollados (Modelo 1 y Modelo 3). Sin embargo, no ha sido una buena práctica hacer *fine-tuning*, ya que el Modelo 2 y el Modelo 4 (los que hacían *fine-tuning*) tienen malos resultados, posiblemente por las diferencias entre los *datasets* usados para el *fine-tuning* y el entrenamiento. Aun así, el Modelo 3 tiene resultados ligeramente mejores que el Modelo 1 por lo que podemos confirmar que estábamos en la dirección correcta al principio del proyecto pensando que las imágenes con personas se describen de manera diferente y que es una buena práctica restringir el entrenamiento a esta clase de imágenes.

8.2 Trabajo a futuro

Considerando cumplidos los objetivos principales de este proyecto, hay varias direcciones en las que se puede enfocar el trabajo a futuro para ampliarlo. Primero de todo, la cantidad y calidad del dato es esencial para un buen entrenamiento, y por tanto el modelo puede ser más exacto y tener mejores resultados de evaluación si tenemos más ejemplos y más representativos en el *dataset* y etiquetados de manera más precisa.

Otra opción es mejorar nuestra capacidad hardware con más GPUs, una GPU mejor o de alguna otra forma en la que podamos obtener más capacidad computacional, para poder entrenar el modelo más rápidamente.

Además de todos estos cambios para mejorar los resultados de nuestros modelos, otra dirección que podemos tomar en el futuro es ampliar el alcance de los objetivos del modelo. Podemos incluir descripciones más completas que reflejen características más complejas de la persona en la imagen, como los sentimientos de la persona (si está sonriendo o está triste), o alguna otra característica que no esté explicíta en la imagen pero que se pueda derivar de ella, sin limitarnos a describir solamente lo que está estrictamente en la imagen. Además, el modelo puede extenderse a la descripción de no solo personas, sino otros objetos que aparezcan en la imagen y la relación espacial entre esos objetos y las personas (por ejemplo: *el hombre detrás de la ventana*, *la señora al lado de la silla azul*).

También hay otros problemas emergentes que combinan visión por computador y procesamiento del lenguaje natural, intentando ir más allá de lo que está en la imagen, que podría ser interesante estudiar como trabajo a futuro de este proyecto; por ejemplo, la tarea de VQA [Wu et al., 2017a], que trata de buscar respuestas a preguntas sobre una imagen, o cuentacuentos visual [Huang et al., 2016, Mostafazadeh et al., 2017], que genera una descripción narrativa de la imagen de entrada, haciendo hipótesis subjetivas de lo que está pasando.

Otra posibilidad interesante es implementar un modelo de machine learning, usando algoritmos del estado del arte (como xgboost), para analizar las diferencias entre los modelos de aprendizaje profundo y machine learning (tiempo computacional, necesidades del *dataset*, precisión, etc).

Por último, aunque las redes neuronales convolucionales son actualmente el estado del

arte, tienen algunas limitaciones que reducen el nivel de precisión del modelo. En los últimos meses, ha aparecido un nuevo tipo de redes neuronales que tratan de resolver estas limitaciones, se llaman redes neuronales cápsula. Podría ser una buena dirección a futuro implementar el problema usando estas redes neuronales y comparar sus resultados con los obtenidos con las convolucionales para ver si obtiene mejores resultados.

8.2.1 Limitaciones de las CNN

Las redes neuronales convolucionales son hasta ahora el estado del arte en el campo del análisis de imágenes, pero tienen algunas limitaciones cuando se trata de relaciones espaciales.

Estas redes se convirtieron en un gran avance comparadas a las redes neuronales tradicionales, ya que permitían que la entrada del modelo fuese una matriz de dos dimensiones a diferencia de la necesidad de aplanar en un vector la matriz de píxeles, lo que, junto con las operaciones de convolución y *pooling*, permitió que los modelos fuesen invariantes respecto a traslaciones.

Como ya hemos comentado a lo largo de este proyecto, el trabajo de las capas convolucionales en una CNN es detectar las características importantes de la imagen, donde las capas más cercanas a la entrada detectan características simples y las capas más profundas combinan estas características para detectar otras más complejas. Por otro lado, las capas de *pooling* ayudan a reducir la dimensionalidad de la imagen, y, por tanto, a reducir en tiempo computacional, y a resumir la información importante de la imagen, así como crear invarianza espacial.

Sin embargo, esta invarianza pierde la relación espacial existente entre todas estas características detectadas. No tiene en cuenta como las diferentes características están relacionadas entre sí ya que se pierde su posición exacta. Esto hace que el modelo pueda dar un falso positivo cuando la imagen tiene todos los componentes de un objeto pero no en el correcto orden. Por ejemplo, puede considerar que es una cara cuando tiene dos ojos, una nariz y una boca pero la boca está donde debe estar un ojo y viceversa.

Además, las redes neuronales convolucionales no tienen en cuenta las características espaciales de cada característica. Esto es, no reconocen un objeto que ya han visto si está en otra orientación. Siguiendo con el ejemplo anterior, una imagen con la cara boca abajo no se reconocería como una cara. Para solucionar esto, los modelos convolucionales se entrenan enseñando imágenes explícitamente a la red desde todos los ángulos posibles.

Sin embargo, la solución no es quitar las capas de *pooling* de los modelos, porque es necesaria la invarianza que introducen; si no, el modelo solo reconocería imágenes muy similares a las del conjunto de entrenamiento. Lo que un buen modelo necesita es ajustar esta invarianza con equivarianza, esto es, entender los cambios de rotación y proporción y adaptarse a ellos.

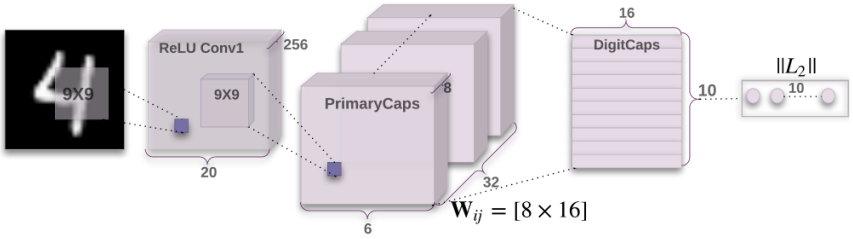


Figure 8.1: Estructura de una red neuronal cápsula (CapsNet).

8.2.2 Redes neuronales cápsula

Geoffrey E. Hinton, conocido como el padre del aprendizaje profundo, ha estado comentando desde hace un tiempo las limitaciones de las redes neuronales convolucionales y la necesidad de sugerir nuevos modelos que las solucionen. En 2011 publicó un artículo [Hinton et al., 2011] con una propuesta de incluir características espaciales más complejas a las redes neuronales, pero no ha sido hasta hace poco que finalmente ha publicado dos artículos [Hinton et al., 2018, Sabour et al., 2017] explicando su nuevo modelo de redes neuronales, las redes cápsula.

En el artículo que han publicado [Sabour et al., 2017], proponen CapsNet, una red neuronal cápsula para el *dataset* MNIST[1], que obtiene mejores resultados que las redes convolucionales y sin la necesidad de hacer *data augmentation*. La Figura 8.1 representa este modelo.

De acuerdo con Hinton, *una cápsula es un conjunto de neuronas cuyas salidas representan diferentes propiedades de una misma entidad*. Son capas anidadas dentro de una capa, cada una de ellas centrada en detectar una característica particular en la imagen y devolviendo como salida un vector que representa la existencia de esa característica y las propiedades de su pose.

La longitud del vector representa la probabilidad de la existencia de la característica y las propiedades espaciales están codificadas en la dirección del vector. De esta forma, cuando la característica detectada se mueve o cambia su estado espacial, la longitud del vector no cambia (la probabilidad se mantiene igual) pero cambia su orientación.

En estas redes neuronales, las cápsulas de capas inferiores deciden dinámicamente cómo enviar su vector de salida a las cápsulas de la siguiente capa. Cada cápsula en la capa inferior calcula, para cada posible padre, un vector de predicción de la pose de la característica de una cápsula de la capa superior (lo que la cápsula superior *vería*), multiplicando su vector de salida por una matriz de pesos. Cuando varias cápsulas de una capa coinciden en lo que pueden haber detectado, activan la cápsula correspondiente de

[1]Un *dataset* de dígitos escritos a mano.

la siguiente capa.

Hinton llama a este método el *algoritmo de enrutación por acuerdo*, y sustituye al algoritmo de *pooling* de las redes convolucionales.

Bibliography

[DBL, 2015] (2015). *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society.

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zhang, X. (2016). Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695.

[Altman, 1992] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

[Baik and Bala, 2004] Baik, S. and Bala, J. W. (2004). A decision tree algorithm for distributed data mining: Towards network intrusion detection. In Laganà, A., Gavrilova, M. L., Kumar, V., Mun, Y., Tan, C. J. K., and Gervasi, O., editors, *Computational Science and Its Applications - ICCSA 2004, International Conference, Assisi, Italy, May 14-17, 2004, Proceedings, Part IV*, volume 3046 of *Lecture Notes in Computer Science*, pages 206–212. Springer.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Gool, L. J. V. (2006). SURF: speeded up robust features. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part I*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer.

[Bengio et al., 1994] Bengio, Y., Simard, P. Y., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166.

[Bernardi et al., 2016] Bernardi, R., Çakici, R., Elliott, D., Erdem, A., Erdem, E., Ikizler-Cinbis, N., Keller, F., Muscat, A., and Plank, B. (2016). Automatic description generation from images: A survey of models, datasets, and evaluation measures. *J. Artif. Intell. Res.*, 55:409–442.

[Bridge et al., 2014] Bridge, J. P., Holden, S. B., and Paulson, L. C. (2014). Machine learning for first-order theorem proving - learning to select a good heuristic. *J. Autom. Reasoning*, 53(2):141–172.

[Canny, 1986] Canny, J. F. (1986). A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698.

[Chen et al., 2015] Chen, X., Fang, H., Lin, T., Vedantam, R., Gupta, S., Dollár, P., and Zitnick, C. L. (2015). Microsoft COCO captions: Data collection and evaluation server. *CoRR*, abs/1504.00325.

[Chen and Zitnick, 2015] Chen, X. and Zitnick, C. L. (2015). Mind's eye: A recurrent visual representation for image caption generation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 2422–2431.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In [Moschitti et al., 2014], pages 1724–1734.

[Chung et al., 2014] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

[Collobert et al., 2011] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Dean et al., 2012] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In *NIPS*.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society.

[Denkowski and Lavie, 2014] Denkowski, M. J. and Lavie, A. (2014). Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL 2014, June 26-27, 2014, Baltimore, Maryland, USA*, pages 376–380. The Association for Computer Linguistics.

[Donahue et al., 2015] Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Darrell, T., and Saenko, K. (2015). Long-term recurrent convolutional networks for visual recognition and description. In [DBL, 2015], pages 2625–2634.

[Duchi et al., 2011] Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

[Goldberg et al., 1994] Goldberg, E., Driedger, N., and Kittredge, R. I. (1994). Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53.

[Graves et al., 2013] Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE.

[Harris and Stephens, 1988] Harris, C. G. and Stephens, M. (1988). A combined corner and edge detector. In Taylor, C. J., editor, *Proceedings of the Alvey Vision Conference, AVC 1988, Manchester, UK, September, 1988*, pages 1–6. Alvey Vision Club.

[Hendricks et al., 2016] Hendricks, L. A., Venugopalan, S., Rohrbach, M., Mooney, R. J., Saenko, K., and Darrell, T. (2016). Deep compositional captioning: Describing novel object categories without paired training data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1–10.

[Hinton et al., 2011] Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In Honkela, T., Duch, W., Girolami, M. A., and Kaski, S., editors, *Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I*, volume 6791 of *Lecture Notes in Computer Science*, pages 44–51. Springer.

[Hinton et al., 2018] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with EM routing. In *International Conference on Learning Representations*.

[Ho, 1995] Ho, T. K. (1995). Random decision forests. In *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*, pages 278–282. IEEE Computer Society.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Hodosh et al., 2013] Hodosh, M., Young, P., and Hockenmaier, J. (2013). Framing image description as a ranking task: Data, models and evaluation metrics. *J. Artif. Intell. Res.*, 47:853–899.

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables with principal components. *Journal of Educational Psychology*, 24:417–441.

[Huang et al., 2016] Huang, T. K., Ferraro, F., Mostafazadeh, N., Misra, I., Agrawal, A., Devlin, J., Girshick, R. B., He, X., Kohli, P., Batra, D., Zitnick, C. L., Parikh, D., Vanderwende, L., Galley, M., and Mitchell, M. (2016). Visual storytelling. In Knight, K., Nenkova, A., and Rambow, O., editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1233–1239. The Association for Computational Linguistics.

[Iordanskaja et al., 1992] Iordanskaja, L., Kim, M., Kittredge, R. I., Lavoie, B., and Polguère, A. (1992). Generation of extended bilingual statistical reports. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*, pages 1019–1023.

[Jia et al., 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

[Karpathy and Li, 2015] Karpathy, A. and Li, F. (2015). Deep visual-semantic alignments for generating image descriptions. In [DBL, 2015], pages 3128–3137.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kiros et al., 2014a] Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014a). Multimodal neural language models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 595–603.

[Kiros et al., 2014b] Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014b). Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, abs/1411.2539.

[Krizhevsky, 2012] Krizhevsky, A. (2012). Learning multiple layers of features from tiny images.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114.

[Kulkarni et al., 2013] Kulkarni, G., Premraj, V., Ordonez, V., Dhar, S., Li, S., Choi, Y., Berg, A. C., and Berg, T. L. (2013). Babytalk: Understanding and generating simple image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(12):2891–2903.

[Kuznetsova et al., 2012] Kuznetsova, P., Ordonez, V., Berg, A. C., Berg, T. L., and Choi, Y. (2012). Collective generation of natural image descriptions. In *The 50th*

Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers, pages 359–368. The Association for Computer Linguistics.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Li et al., 2011] Li, S., Kulkarni, G., Berg, T. L., Berg, A. C., and Choi, Y. (2011). Composing simple image descriptions using web-scale n-grams. In Goldwater, S. and Manning, C. D., editors, *Proceedings of the Fifteenth Conference on Computational Natural Language Learning, CoNLL 2011, Portland, Oregon, USA, June 23-24, 2011*, pages 220–228. ACL.

[Lin, 2004] Lin, C.-Y. (2004). Rouge: a package for automatic evaluation of summaries.

[Lin et al., 2014] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157.

[Mao et al., 2014] Mao, J., Xu, W., Yang, Y., Wang, J., and Yuille, A. L. (2014). Deep captioning with multimodal recurrent neural networks (m-rnn). *CoRR*, abs/1412.6632.

[Mason and Charniak, 2014] Mason, R. and Charniak, E. (2014). Nonparametric method for data-driven image captioning. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 592–598. The Association for Computer Linguistics.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

[Mitchell et al., 2012] Mitchell, M., Dodge, J., Goyal, A., Yamaguchi, K., Stratos, K., Han, X., Mensch, A., Berg, A. C., Berg, T. L., and III, H. D. (2012). Midge: Generating image descriptions from computer vision detections. In Daelemans, W., Lapata, M., and Màrquez, L., editors, *EACL 2012, 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France, April 23-27, 2012*, pages 747–756. The Association for Computer Linguistics.

[Moschitti et al., 2014] Moschitti, A., Pang, B., and Daelemans, W., editors (2014). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL.

[Mostafazadeh et al., 2017] Mostafazadeh, N., Brockett, C., Dolan, B., Galley, M., Gao, J., Spithourakis, G. P., and Vanderwende, L. (2017). Image-grounded conversations:

Multimodal context for natural question and response generation. In Kondrak, G. and Watanabe, T., editors, *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017 - Volume 1: Long Papers*, pages 462–472. Asian Federation of Natural Language Processing.

[Mozer, 1989] Mozer, M. C. (1989). A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3(4).

[Ordonez et al., 2011] Ordonez, V., Kulkarni, G., and Berg, T. L. (2011). Im2text: Describing images using 1 million captioned photographs. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 1143–1151.

[Pang et al., 2002] Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, pages 311–318. ACL.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In [Moschitti et al., 2014], pages 1532–1543.

[Reiter and Dale, 1997] Reiter, E. and Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Russell et al., 2008] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173.

[Sabour et al., 2017] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3859–3869.

[Sarikaya et al., 2014] Sarikaya, R., Hinton, G. E., and Deoras, A. (2014). Application of deep belief networks for natural language understanding. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 22(4):778–784.

[Seide and Agarwal, 2016] Seide, F. and Agarwal, A. (2016). Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2135–2135, New York, NY, USA. ACM.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[Tanti et al., 2017a] Tanti, M., Gatt, A., and Camilleri, K. P. (2017a). What is the role of recurrent neural networks (rnns) in an image caption generator? In *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*, pages 51–60.

[Tanti et al., 2017b] Tanti, M., Gatt, A., and Camilleri, K. P. (2017b). Where to put the image in an image caption generator. *CoRR*, abs/1703.09137.

[Theano Development Team, 2016] Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

[Tran et al., 2016] Tran, K., He, X., Zhang, L., and Sun, J. (2016). Rich image captioning in the wild. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016, Las Vegas, NV, USA, June 26 - July 1, 2016*, pages 434–441. IEEE Computer Society.

[Vedantam et al., 2014] Vedantam, R., Zitnick, C. L., and Parikh, D. (2014). Cider: Consensus-based image description evaluation. *CoRR*, abs/1411.5726.

[Vinyals et al., 2015] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In [DBL, 2015], pages 3156–3164.

[Wernick et al., 2010] Wernick, M. N., Yang, Y., Brankov, J. G., Yourganov, G., and Strother, S. C. (2010). Machine learning in medical imaging. *IEEE Signal Processing Magazine*, 27(4):25–38.

[Wu et al., 2017a] Wu, Q., Teney, D., Wang, P., Shen, C., Dick, A. R., and van den Hengel, A. (2017a). Visual question answering: A survey of methods and datasets. *Computer Vision and Image Understanding*, 163:21–40.

[Wu et al., 2017b] Wu, S., Wieland, J., Farivar, O., and Schiller, J. (2017b). Automatic alt-text: Computer-generated image descriptions for blind users on a social network service. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, pages 1180–1192, New York, NY, USA. ACM.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

[Yang et al., 2011] Yang, Y., Teo, C. L., III, H. D., and Aloimonos, Y. (2011). Corpus-guided sentence generation of natural images. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 444–454. ACL.

[Young et al., 2014] Young, P., Lai, A., Hodosh, M., and Hockenmaier, J. (2014). From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *TACL*, 2:67–78.